



Mala škola programiranja C#

Microsoft®

Mala škola programiranja C# (1)

Uvod

Namena ovog serijala jeste da nastavnike informatike i programiranja uputi u tematiku programiranja u programskom jeziku C# i u .NET tehnologiju.

Kroz niz primera kojima prethodi teorijska osnova biće obrađeno više različitih aspekata i tipova aplikacija koje se mogu realizovati, što uključuje konzolne i Windows aplikacije, kreiranje biblioteka klasa i pristup Microsoft SQL Server bazi podataka.

Platforma u kojoj se radi jeste Microsoft Visual C# 2005 Express Edition i Microsoft SQL Server 2005 Express Edition. Odlučili smo da koristimo express izdanja jer su besplatna i mogu se preuzeti sa Microsoft Web lokacije. Naravno, svi primeri će ispravno funkcionisati i u bilo kojoj drugoj verziji programa Microsoft Visual Studio 2005.

Express alatke

Microsoft je kreirao niz besplatnih, express izdanja svojih razvojnih alatki i SQL Server baze podataka sa idejom da svima omogući učenje, testiranje i kreiranje aplikacija u .NET framework okruženju. Ovo ne znači da se u express alatkama ne mogu napraviti ozbiljne aplikacije, Web lokacije i baze podataka. Naprotiv, dostupna je većina mogućnosti koje postoje i u kompletnoj verziji programa Visual Studio, tako da se mogu kreirati snažne i bogate aplikacije i sistemi sa bazama podataka.

Matična stranica za sve Express alatke je:
<http://msdn.microsoft.com/vstudio/express/>. Web stranica sa koje možete preuzeti C# Express je:
<http://msdn.microsoft.com/vstudio/express/visualcsharp/download/>.

Instalaciju možete izvršiti sa Interneta ili preuzeti sadržaj kompletног instalacionog CD-a. Druga opcija je bolja, jer kada jednom preuzmete instalaciju možete je kopirati na onoliko diskova koliko vam je potrebno za školu, učenike i nastavnike.

Kada otvorite navedenu stranicu, kliknite na vezu „*uputstva za ručnu instalaciju*“ (*manual installation instructions*) koja vas vodi na lokaciju gde možete preuzeti kompletну instalaciju u IMG ili ISO formatu. Svi popularni programi za kopiranje kompakt diskova podržavaju oba formata. Veličina instalacije je približno 435 MB.

File Packaging and Download Paths

Each Express Edition is packaged in its entirety, including all optional components, as an image (img or iso) file. Below are links to the individual image files for each Express Edition.

Product	Total Size	Download	CRC Number
Visual Web Developer 2005 Express Edition	449,848 KB	.IMG File .ISO File	F972C10F
Visual Basic 2005 Express Edition	445,282 KB	.IMG File .ISO File	BAC91B78
Visual C# 2005 Express Edition	445,282 KB	.IMG File .ISO File	55884F2C
Visual C++ 2005 Express Edition	474,686 KB	.IMG File .ISO File	3DE23D4A
Visual J# 2005 Express Edition	448,702 KB	.IMG File .ISO File	91B03EAS

Tehnologija

Microsoft je predstavio .NET tehnologiju 2000. godine, uz prvu verziju programa Visual Studio 2000 i .NET Framework 1.0. Kasnije je usledila verzija programa 2003 Visual Studio i 1.1 .NET Framework. Trenutno su aktuelne verzije 2005 Visual Studio, odnosno 2.0 .NET Framework.

.NET Framework predstavlja obiman skup klasa koje se koriste u programiranju različitih tipova aplikacija. Sve klase su grupisane u takozvane imenovane prostore (named spaces) radi lakšeg snalaženja i efikasnijeg rada. Na primer, klase za rad sa SQL bazom podataka nalaze se u imenovanom prostoru System.Data.SqlClient, klase za rad sa datotekama nalaze se u prostoru System.IO i tako dalje. Sve što vam ikada može zatrebatи gotovo sigurno se može pronaći ovde. Ako ne postoji, može se napisati od početka ili naslediti funkcionalnost postojećih klasa.

Nepravedno je govoriti o C#, a ne pomenuti *Anders Hejlsberg*, koji je glavni arhitekta i projektant ovog jezika i učesnik u razvoju .NET tehnologije. Rođen u Danskoj 1961. godine, programer koji je početkom osamdesetih godina 20. veka napisao čuveni Turbo Pascal, a kasnije projektovao Delphi. 1996. godine Anders se pridružio korporaciji Microsoft gde je i razvio programski jezik J++ i biblioteku klasa „Windows Foundation Classes“.

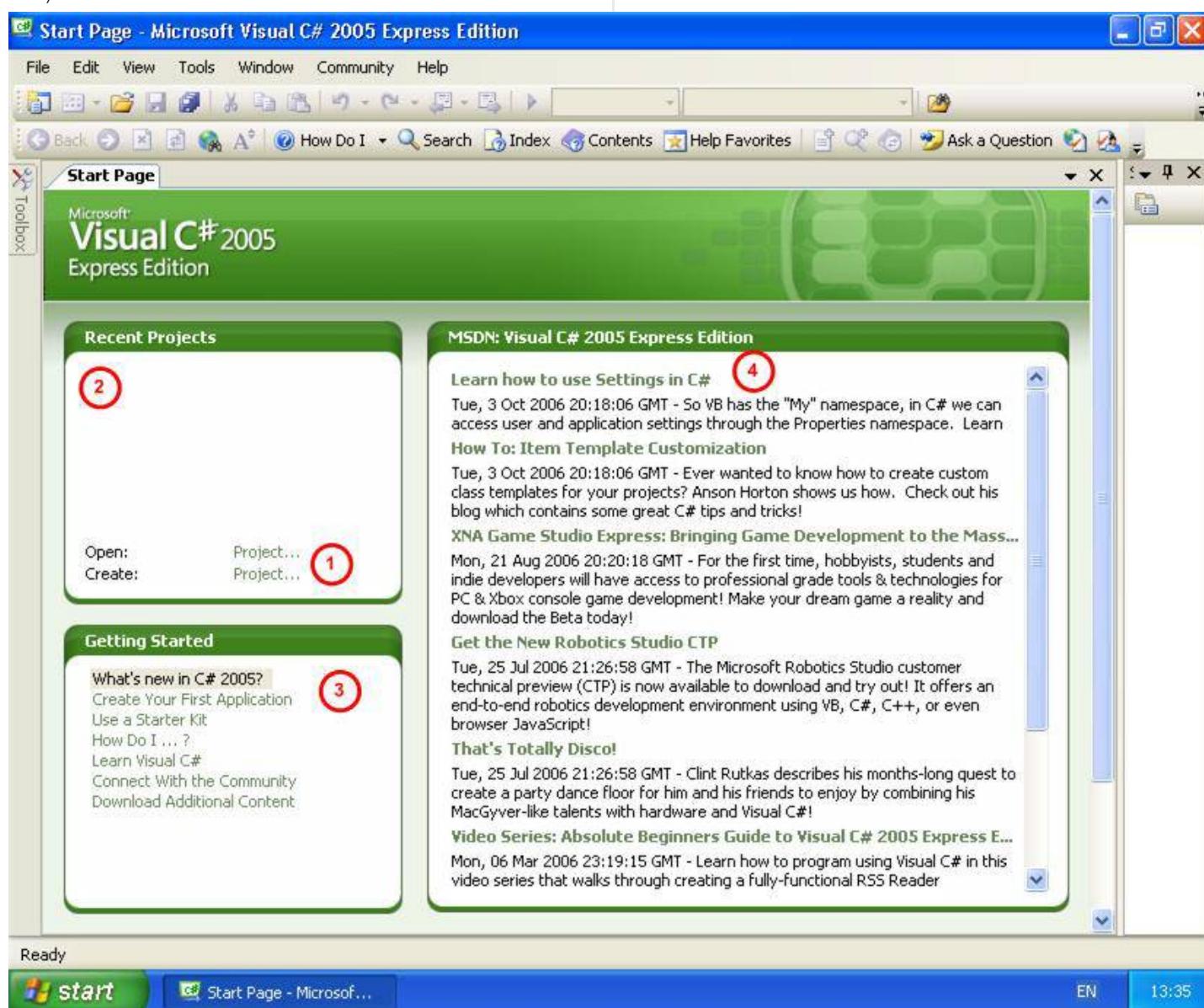
.NET programski jezici mogu pristupiti svim ovim klasama i u tom smislu su prilično ravnopravni, bez obzira na to da li koristite Visual Basic .NET, C#, Fortran .NET ili neki drugi programski jezik. Razlika između jezika svodi se na sintaksu, strukturu i lični izbor. Trenutno su svi najčešće korišćeni programski jezici prilagođeni .NET tehnologiji.

Programski jezik C# se izdvaja od drugih i smatra se prirodnim za .NET okruženje. Ovaj programski jezik je nastao kao potpuno nov i nije opterećen kompatibilnošću sa ranijim verzijama. Radno okruženje Visual Studio je u potpunosti napisano baš u C# jeziku. On je moderno strukturiran, potpuno objektno orijentisan, zasnovan na C++ jeziku i zvanično prihvaćen kao standard od strane organizacija ECMA (*European Computer Manufacturers Association*) i ISO (*International Organization for Standardization*).

.NET prevodenje

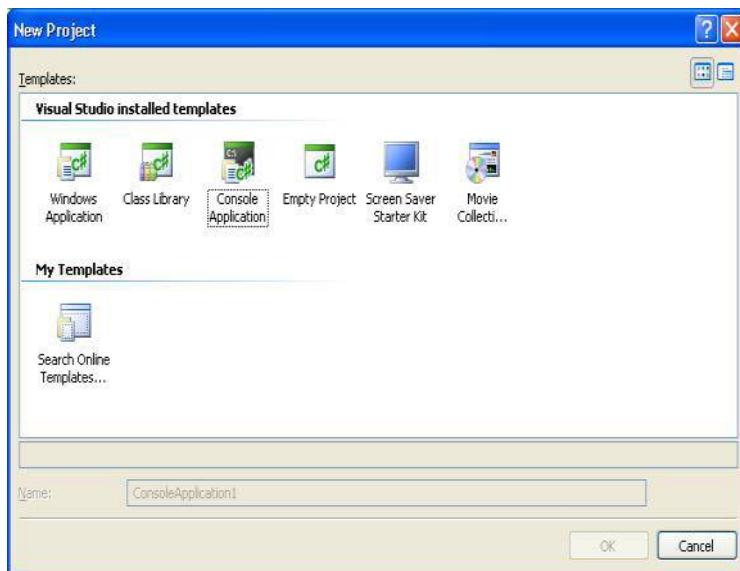
Prilikom prevodenja koda napisanog u bilo kojem programskom jeziku postoje dve mogućnosti: prevodioци i tumači.

Prevodioци prevode izvorni kôd direktno u mašinski kôd procesora. U slučaju tumača se prilikom pokretanja programa prevodi linija po linija - kako se program izvršava. NET je tumač, ali ne vrši prevodenje direktno u mašinski jezik procesora, već u takozvani MSIL (Microsoft Intermediate Language). Po pokretanju programa na scenu stupa JIT (Just In Time) prevodilac, koji dalje prevodi MSIL u mašinski jezik procesora i izvršava ga. Ovakva slojedjiva arhitektura omogućava portabilnost .NET aplikacija između različitih operativnih sistema u budućnosti.



Početak

Za početak je potreban operativni sistem Windows XP i instaliran C# Express 2005. Kada se pokrene radno okruženje, na uvodnom ekranu postoje opcije za kreiranje novog ili otvaranje postojećeg projekta (1). Takođe, tu se nalazi i lista prethodno otvorenih projekata (2) koja je sada prazna. Veoma su korisna i okna sa vezama za početnike (3) i najnovijim vestima (4) koje se automatski ažuri-



ra ako postoji Internet veza.

Potrebno je kliknuti na vezu *Kreiraj: Projekat...* (*Create: Project...*), čime se otvara dijalog u kome treba izabrati tip projekta kao što je prikazano na prethodnoj slici.

Tri osnovna tipa projekta su:

- Windows aplikacija (Windows application) predstavlja standardnu Windows aplikaciju koja se može samostalno pokretati na računaru.
- Biblioteka klase (Class Library) je biblioteka klasa sa svojim funkcijama i drugim elementima. Ovaj projekat ne može se samostalno pokrenuti, već ga pokreće i koriste drugi tipovi projekata.
- Aplikacija konzole (Console Application) takođe Windows aplikacija, ali bez grafičkog korisničkog interfejsa. Komunikacija se odvija isključivo sa komandne linije.

Ostali tipovi predstavljaju samo različite predloške i demo aplikacije uz mogućnost pronalaženja drugih predložaka projekata na Internetu.

Za početak, kao najjednostavniji tip aplikacije, treba

izabrati stavku „Aplikacija konzole“ (Console Application) i kliknuti na dugme „U redu“ (OK).

Sve linije primera koda u ovom članku su prikazane crvenom bojom, a kôd koji je automatski generisalo radno okruženje prikazan je sivom bojom.

Posle par momenata se otvara prozor za pisanje koda sa već ispisanim kosturom aplikacije. Potrebno je obratiti pažnju na liniju `static void Main(string[] args)`. Ona definiše glavnu (Main) funkciju koja se prva pokreće prilikom pokretanja programa. U telu ove funkcije treba upisati kôd koji je potrebno izvršiti. Telo funkcije je definisano početnom { i završnom } velikom zagradom, što predstavlja način na koji se obeležava početak i kraj svih segmenta u C# jeziku.

Da biste probali jednostavan ispis na ekranu, u okviru velikih zagrada upišite tekst tačno kao što je prikazano:

```
class Program
```

```
{
```

```
    static void Main(string[] args)
```

```
{
```

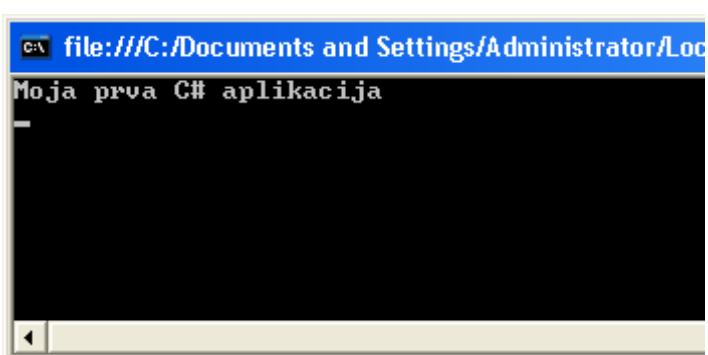
```
    Console.WriteLine ("Moja prva C# aplikacija");
```

```
    Console.ReadKey ();
```

```
}
```

Imajte na umu da C# razlikuje mala i velika slova!

Pokretanje aplikacije iz radnog okruženja vrši se pomoću funkcijskog tastera F5 na tastaturi. Ako je kôd ispravno napisan, pojavljuje se prozor komandne linije sa ispisanim tekstom:



Pritisnite bilo koji taster čime se zatvara prozor komandne linije, odnosno zaustavlja aplikacija.

Analiza koda:

Linija: `Console.WriteLine("Moja prva C# aplikacija");`
je „odgovorna“ za ispis teksta. `Console` objekt predstavlja prozor komandne linije, dok je `WriteLine` metod ovog objekta kojim se ispisuje tekst. Kasnije ćemo videti šta su klasa, objekat i metoda klase. Očigledno, parametar metode `WriteLine` je tekst koji želimo da se ispiše.

Linija: `Console.ReadKey();`

postoji samo da bi se napravila pauza. Bez nje bi se otvorio prozor komandne linije, ispisao tekst i prozor bi se **odmah zatvorio jer se aplikacija završila**. Slično kao i u prethodnoj liniji koristi se objekat `Console`, a ovog puta metod `ReadKey` koji čeka da korisnik pritisne taster na tastaturi. *Pokušajte da obrišete ovu liniju i pokrenete program i vidite šta se događa.*

Za pojmove: klasa, objekat, objektno programiranje, nasleđivanje, metodi i slično čuli ste ili pročitali u literaturi i oni (po mom iskustvu) izazivaju odbojnost ka objektnom programiranju, bez obzira na to da li ste početnik ili profesionalac. Ne treba se tako osećati jer objektno programiranje predstavlja način razmišljanja koje je najbliže svakodnevnom iskustvu i najuspešnije modelira stvarnost.

U sledećem nastavku serijala o programiranju bavćemo se baš ovim pojmovima jer oni predstavljaju okosnicu u C# programiranju.

Mala škola programiranja C# (2)

U ovom nastavku male škole jezika C# posvetićemo se osnovama objektnog programiranja. Za razliku od tradicionalnog proceduralnog programiranja koje se sastoji od niza funkcija i njihovih poziva, objektno programiranje bolje i skladnije oslikava realni svet i omogućava logičnije projektovanje programske zahteve. Da odmah u početku bude jasno: problem se može rešiti pomoću tradicionalnog i objektnog programiranja.

Ono što nam donosi objektno programiranje je skladniji programski model i još neke lepe stvari kao što je ponovna upotrebljivost i nasleđivanje objekata.

Jedan objekat je uvek predstavljen jednom klasom. Klasu treba posmatrati kao nacrt ili plan za izgradnju objekta. Na primer, plan za izgradnju kuće predstavlja klasu, a kuća napravljena po tom planu predstavlja objekat zasnovan na klasi. Sama klasa za sebe nije objekat, već predstavlja samo opis po kome će objekat biti napravljen. Kada napravimo (kaže se „instanciramo“) objekat na osnovu klase, on se kreira u memoriji računara i tek tada se može koristiti. Na osnovu jedne klase može se napraviti neograničeni broj objekata, odnosno broj koji je tehnički ograničen memorijom računara.

Svojstva, metode i događaji

Uzmimo učenika kao primer. Ako bismo hteli da ga predstavimo u objektnom svetu, šta bi klasa koja ga opisuje trebalo da sadrži? Možemo početi od fizičkih karakteristika. Ime učenika, pol, godina rođenja, boja očiju i kose, razred i slično. Sve pomenuto predstavlja svojstva (eng. properties) klase. Kada instanciramo objekat na osnovu klase „Učenik“, postavićemo ova svojstva da odgovaraju konkretnom učeniku.

Očigledno, na osnovu dobro kreirane klase „učenik“ možemo tačno opisati sve učenike svoje škole. Međutim, ovo nije dovoljno. Učenik obavlja veliki broj školskih aktivnosti koje takođe treba opisati. To je ono što učenik radi u školi i kod kuće, na primer, radi pismeni i domaći zadatki, prisustvuje određenom času, dobija ocenu i slično. Ove radnje koje učenik izvršava i koje se odnose na njega nazivaju se metodima (eng. methods) klase.

Učenik takođe na određeni način reaguje na dešavanja u školskom okruženju. Reakcija na dobru ili lošu ocenu, izostanak u slučaju bolesti i slično predstavljaju događaje (eng. events) klase.

Sada, posle detaljne analize svih osobina, radnji i reakcija učenika moguće je kreirati klasu koja verno i detaljno može da opiše učenika sa svim potrebnim karakteristikama.

Fizička implementacija klase

Svojstva klase su predstavljena različitim varijablama koje su javno vidljive van klase. Korisnik klase će po kreiranju objekta na osnovu klase uspostaviti odgovarajuće vrednosti varijabli i time definisati objekat.

Metode klase su predstavljene funkcijama u klasi. Na primer, funkcija „UradiDomaćiZadatak“ koja bi kao ulazne parametre verovatno trebalo da sadrži opis domaćeg zadatka, predmet, vreme kada je zadat i rok izvršavanja. Metod klase može imati i povratnu vrednost koja se obično koristi kao status izvršavanja metoda, odnosno funkcije. Nepisano pravilo jeste da povratna vrednost nula znači uspešno izvršenu radnju, a vrednost različita od nule označava problem definisan tim brojem.

Da bismo mogli da napravimo prvi primer za demonstraciju jednostavne klase, neophodno je da se upoznamo sa načinom deklaracije varijabli i funkcija u C# jeziku.

Deklaracija varijabli i funkcija

Sve varijable u programskom jeziku C# moraju se deklarisati pre upotrebe. Deklaracija varijable podrazumeva da definišemo naziv, tip, vidljivost i optionalno njenu početnu vrednost. Sintaksa je sledeća:

vidljivost tip varijable naziv = početna vrednost

Na primer:

```
public string ImeUcenika;
private int Razred = 1;
```

U prvom primeru je deklarisana varijabla pod nazivom „ImeUcenika“ koja pripada tipu „niska“ (eng. string), što znači da može da sadrži bilo koji niz znakova i njena vidljivost je javna (public).

Vidljivost definiše odakle se varijabli može pristupiti. Ključna reč „public“ znači da je varijabla javna u celom opsegu u kome je deklarisana – na primer, u celoj klasi i da je takođe vidljiva korisnicima te klase.

U drugom slučaju deklarisali smo varijablu pod nazivom „Razred“ koja pripada tipu „int“ (pozitivni i negativni celi brojevi u određenom opsegu) sa privatnom vidljivošću – dostupna je samo u okviru dela u kome je deklarirana, ali ne i korisnicima klase. Inicijalnu vrednost variable takođe smo postavili na 1.

Osim javnih (eng. public) i privatnih (eng. private) tipova vidljivosti postoji još nekoliko varijanti o kojima će biti govora kasnije. Za početak je dovoljno znati da privatne variable nisu vidljive van klase, a javne jesu.

Deklaracija funkcija je slična. Takođe se definije vidljivost, ime funkcije, tip (koji u slučaju funkcije znači tip vrednosti koju vraća funkcija) uz dodatak deklaracije ulaznih argumenata funkcije ako postoje.

Primer funkcije koja vraća zbir dva broja:

```
public double Zbir (double a, double b)
{
    return a + b;
}
```

Funkcija Zbir je javna (dostupna korisnicima klase) i kao rezultat vraća tip „double“ (broj sa decimalnom tačkom). Postoje dva ulazna parametra a i b koji su takođe tipa „double“. Telo funkcije je određeno otvorenom i zatvorenom vitičastom zagradom. Naredba return označava bezuslovni završetak rada funkcije i povratnu vrednost funkcije. U našem primeru funkcije postoji samo naredba return koja vraća zbir dva broja.

U složenijim klasama može postojati puno funkcija koje implementiraju različite metode klase. Neke od njih će imati povratnu vrednost, a neke ne. Funkcije koje nemaju povratnu vrednost deklarišu se kao tip void. Na primer:

```
public void PokreniProgram (string LokacijaPrograma)
{
    // kôd koji pokreće program...
    return;
}
```

U tom slučaju se navodi samo ključna reč return bez parametra. U primeru je takođe prikazan način za upisivanje komentara u programskom jeziku C#. Dve kose crte // označavaju komentar sve do kraja reda.

Ovaj komentar se zanemaruje prilikom prevođenja, a dobra praksa je pisati komentare uvek kada je to potrebno radi boljeg razumevanja koda.

Osim ove jednolinijske varijante komentara postoji i višelinijijski komentar koji počinje parom znakova /* i završava sa */. Sve linije između smatraju se komentarom i ne izvršavaju se.

Primer klase

Sada znamo dovoljno da bismo mogli da kreiramo jednostavnu klasu kao primer. Sama deklaracija klase je jednostavna:

Class NazivKlase

```
{
    /*
        telo klase sa svim potrebnim svojstvima,
        metodima i događajima
    */
}
```

Kao i u prethodnom nastavku primer čemo zasnovati na konzolnoj aplikaciji. Pokrenite C# Express, kliknite na vezu Kreiraj projekat... (eng. Create project...), izaberite tip projekta Konzolna aplikacija (eng. Console Application) i kliknite na dugme U redu (eng. OK).

Zadatak je napraviti klasu „Matematika“ koja će znati da sabere i oduzme dva broja. Klasa će sadržati dva metoda (funkcije) – „Zbir“ i „Razlika“. Obe funkcije imaju po dva ulazna parametra i brojeve koji se sabiraju, odnosno oduzimaju. Svi ulazni parametri i obe funkcije su tipa „double“.

U kodu koji vidite na ekranu prvo treba deklarisati klasu „Matematika“. Deklaracija se upisuje posle deklaracije klase „Program“ kao što je prikazano:

```
class Program
{
    static void Main(string[] args)
    {
    }
}

class Matematika
{
}
```

Sada, u okviru klase „Matematika“ treba napisati funkcije „Zbir“ i „Razlika“:

```
class Matematika
{
    public double Zbir(double A, double B)
    {
        return A + B;
    }

    public double Razlika(double A, double B)
    {
        return A - B;
    }
}
```

Obe funkcije su javne kako bi bile vidljive korisnicima klase, imaju po dva ulazna parametra tipa „double“ i vraćaju zbir, odnosno razliku. Terminološki, klasa „Matematika“ implementira interfejs sa dva metoda čiji su potpis:

```
public double Zbir (double, double) i
public double Razlika (double, double)
```

Klasa je za sada gotova, prelazimo na njeno korišćenje. Kao što je rečeno, klasa sama za sebe ne može da se koristi, već je neophodno napraviti objekat na osnovu nje.

Tehnički gledano, pravljenje objekta se svodi na kreiranje mini-programa u memoriji računara koji sadrži kompletan kôd iz klase. Instanca klase se jednostavno pravi.

[NazivKlase NazivVarijable = new NazivKlase\(\);](#)

U našem primeru klasu pravimo u delu static void Main kao što je prikazano:

```
static void Main(string[] args)
{
    Matematika M = new Matematika();
}
```

Naredba new Matematika() pravi instancu klase „Matematika“ - objekat u memoriji računara. Varijabla M je pokazivač na memoriju lokaciju na kojoj se nalazi objekat, a pomoću nje se upućuje na sva svojstva i metode u njemu.

Sintaksa upućivanja je jednostavna: NazivVarijable.NazivMetode (ulazni argumenti).

Sada treba testirati obe metode i ispisati njihov povratni rezultat. Za ovo ćemo, kao i u prošlom nastavku koristiti metod „WriteLine“ ugrađenog objekta „Console“. Brojevi koje prosleđujemo kao ulazne parametre izabrani su potpuno proizvoljno.

Dopunite kôd sledećim redovima:

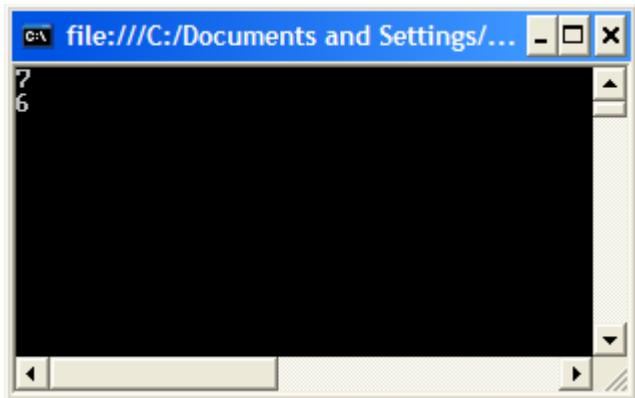
```
static void Main(string[] args)
{
    Matematika M = new Matematika();
    Console.WriteLine ( M.Zbir (3, 4) );
    Console.WriteLine ( M.Razlika (8, 2) );
    Console.ReadKey();
}
```

M.Zbir (3, 4) poziva funkciju „Zbir“ u objektu na koji pokazuje varijabla M i prosleđuje joj vrednosti parametara. Funkcija „Zbir“ kao povratnu vrednost vraća zbir ova dva broja (return 3 + 4). Ovu povratnu vrednost zatim koristimo za ispis u konzolni prozor.

Isti princip važi za pozive funkcije „Razlika“.

Red Console.ReadKey(); služi samo kao veštačka pauza kako se konzolni prozor ne bi odmah zatvorio.

Pokrenite primer pritiskom na funkcijski taster F5.
Ako je kôd dobro unet, dobija se rezultat kao na slici:



Ako je prijavljena greška i primer nije pokrenut, proverite sve redove koda posebno vodeći računa o velikim i malim slovima. Kao što je ranije rečeno, C# razlikuje velika i mala slova tako da su za njega, na primer funkcije Zbir i zbir dve različite funkcije.

Vezbanje

Na osnovu primera u ovom članku, dodajte još dva metoda klasi „Matematika“ koji izvršavaju proizvod i deljenje. Zatim u kodu ispitajte funkcionisanje ova dva nova metoda.

U sledećem nastavku biće obrađena svojstva, kao i druge mogućnosti klase.

Mala škola programiranja C# (3)

Klase - nastavak

Vreme je da naučimo više o svojstvima klase. U prošlom nastavku smo se upoznali sa funkcijama i na koji način se pomoću njih implementiraju metode klase. Svojstva klase, prenesena u realni svet bi mogla da se poistovete sa osobinom objekta ili osobe u prirodi. Na primer, svaki učenik ima niz osobina (svojstava) koje ga opisuju. Ime, godina rođenja, adresa stanovanja, telefon, boja kose i očiju, u kom je razredu, koji smer i tome slično.

U programiranju, kao što smo rekli ranije, klasa predstavlja šablon za objekat. Pošto definišemo svojstva u klasi i na osnovu nje napravimo (instanciramo) više objekata, tipično će svaki od njih imati svoj skup svojstava, baš kao što svaki učenik ima samo svoje osobine.

Programski jezik C# nudi dva osnovna načina za implementaciju svojstava klase.

Implementacija svojstava klase- javne varijable

Najjednostavniji način, od koga ćemo i početi se zasniva na deklaraciji javne (public) varijable unutar klase. Sve što je u klasi definisano kao public, je vidljivo korisnicima ove klase i mogu bez ograničenja postavljati i čitati vrednost ovih varijabli.

Kao i do sada, radićemo sa projektom tipa "konzolna aplikacija" (eng. Console Application). Dakle pokrenite C# Express i izaberite link za novi projekt ovog tipa. Kao i do sada, na ekranu se po šablonu prikazuje početni kod sa ulaznom funkcijom static void Main.

Zadatak je da napravimo klasu Ucenik sa (za sada) svojstvima Prezime i Ime.

Kao i u prošlom nastavku deklaraciju klase treba uraditi posle funkcije static void Main.

```
class Program
{
    static void Main(string[] args)
    {
    }

}

class Ucenik
{
```

Zatim u okviru klase Ucenik, treba uraditi deklaraciju svojstava, odnosno javnih (public) varijabli, obe tipa string:

```
class Ucenik
{
    public string Prezime;
    public string Ime;
}
```

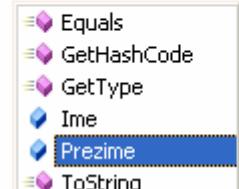
Za sada, veoma jednostavno i očigledno. Kako se koriste ova svojstva na strani korisnika ove klase? U okviru Main funkcije treba prvo deklarisati varijablu koja pokazuje na objekat zasnovan na klasi Ucenik, kao što smo radili ranije. Ime varijable – cu, je proizvoljno izabrano. Postavicemo svojstva Prezime i Ime na proizvoljne vrednosti i potom ispisati sadržaj:

```
class Program
{
    static void Main(string[] args)
    {
        Ucenik cu = new Ucenik();
        cu.Prezime = "Perić";
        cu.Ime = "Petar";

        Console.WriteLine(cu.Prezime);
        Console.WriteLine(cu.Ime);
        Console.ReadKey();
    }
}
```

Obratite pažnju na pomoć koju vam pruža Visual Studio radno okruženje prilikom unosa koda. Kada ste ukucali varijablu cu i potom tačku, automatski se otvorio mali prozor koji vam nudi izbor iz liste svih svojstava i metoda koje poseduje klasa Ucenik:

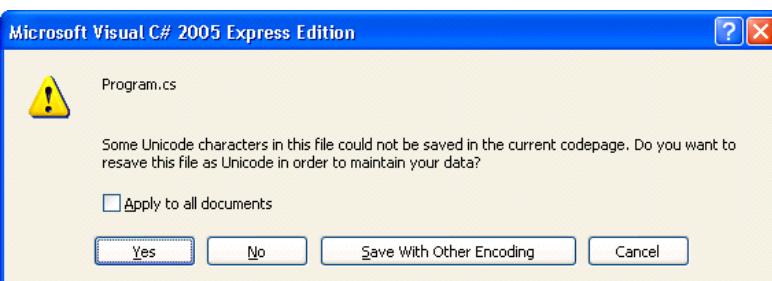
```
static void Main(string[] args)
{
    Ucenik cu = new Ucenik();
    cu.
```



Equals
GetHashCode
GetType
Ime
Prezime
ToString

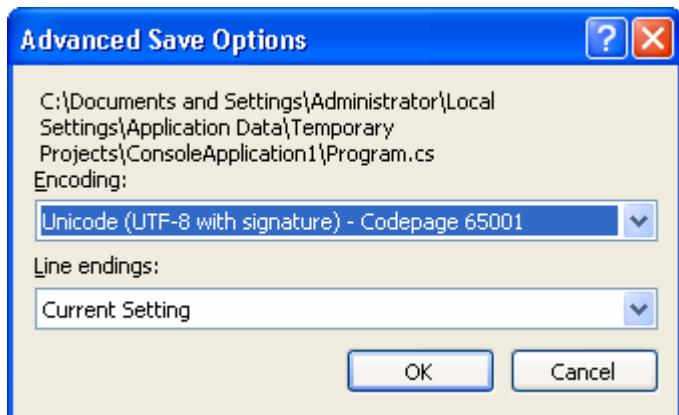
Tehnologija koja pruža ovakvu (i još neke) vrste pomoći se zove IntelliSense i izuzetno je korisna jer skraćuje vreme unosa koda i eliminiše sintaksne greške. Pritisnite da su svojstva prikazana plavom, a metode crvenom bojom. Takođe ćete videti metode koji niste sami napisali. O ovim metodama će biti reči kasnije.

Kada se pokrene ovaj primer (funkcijski taster F5), dobićemo poruku koju do sada nismo imali prilike da vidi-mo:



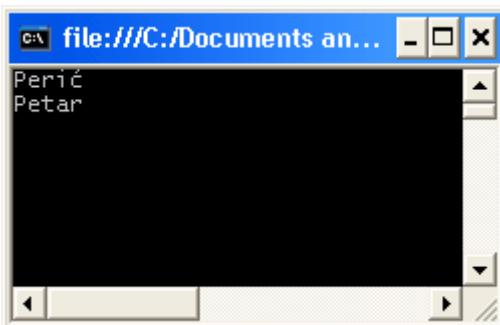
O čemu se radi? Pošto smo u kodu koristili naša latinična slova i to u liniji **cu.Prezime = "Perić";** razvojno okruženje nam postavlja pitanje u kom formatu želimo da sačuvamo izvorni kod kako se ne bi izgubili naši karakteri.

Preciznije rečeno, čim se u kodu nalazi bar jedan karakter van osnovnog ASCII skupa, odnosno latinica, cirilica, slova grčke azbuke i slično, dobićemo isto upozorenje. Da se prezime **Perić** ne bi pretvorilo u Peric, treba kliknuti na dugme "**Sačuvaj sa drugim kodiranjem**" (eng. **Save With Other Encoding**). U narednom dijalogu treba izabrati opciju kao što je prikazano na slici:



Kodiranje "Unicode (UTF-8 with signature)" će sačuvati izvorni kod u univerzalnom slovnom rasporedu koji podržava sve svetske azbuke, pa i našu latinicu i cirilicu. Kliknite na dugme "U redu" (eng. OK) kako bi ispravno snimili kod i više nećete dobijati ovakvo upozorenje.

Rezultat izvršavanja programa je očekivan. Ispisane su vrednosti svojstava Prezime i Ime kao što je prikazano na slici:



Napomena:

Ako umesto "**Perić**" na svom računaru vidite "Peric", nema razloga za brigu. U pitanju je podešavanje fonta koji se koristi za komandni prozor. U svakom slučaju svojstvo

Pritisnite bilo koji taster za završetak programa.

Zaštićena svojstva

Oba svojstva u primeru, korisnik klase može da postavi na bilo koju vrednost. U ovom slučaju to ne predstavlja problem. Ali ako želimo da u klasu Ucenik dodamo svojstvo Razred, pod predpostavkom da je u pitanju učenik osnovne škole, validne vrednosti mogu biti samo brojevi od 1 do 8. Ako svojstvo Razred napravimo kao javnu (public) varijablu, nemamo načina da kontrolišemo i provjerimo vrednost koja je upisana u nju i da zabranimo pogrešne vrednosti.

Da bi ovo ostvarili, moramo posedovati neki mehanizam koji proverava vrednost pre nego što se dodeli varijabli. Ovo se postiže pomoću specijalnih set i get funkcija.

Tehnika je sledeća: umesto da se pravi javna (public) varijabla Razred, treba napraviti privatnu (private) varijablu i radi bolje čitljivosti joj dati slično ime – na primer prRazred (pr od private).

U ovom momentu još uvek nemamo svojstvo Razred, ali ga dodajemo pomoću set i get funkcija. Sintaksa je jednostavna, navodi se tip i naziv svojstva koje je sada javno (public) i u okviru njega upišu set i get funkcije.

Izmeničemo postojeći kod kako bi bilo jasnije:

```

class Ucenik
{
    public string Ime;
    public string Prezime;
    private int prRazred;

    public int Razred
    {
        get
        {
            return prRazred;
        }

        set
        {
            prRazred = value;
        }
    }
}

```

Objašnjenje:

Privatna varijabla prRazred nije vidljiva van klase, i korisnici klase ne mogu niti da postavljaju niti da čitaju njenu vrednost. Ipak, ova varijabla je ona koja je nama bitna i sa kojom radimo unutar svoje klase.

Korisnik može da postavlja vrednost ove varijable, ali na posredan način pomoću public svojstva deklarisanog u liniji `public int Razred`. Svojstvo Razred se sastoji od dva dela: get deo koji se automatski poziva kada korisnik čita vrednost svojstva Razred i set deo koji se automatski poziva kada korisnik postavlja vrednost svojstva Razred.

U get delu, se korisniku zapravo vraća vrednost privatne varijable prRazred:

```
return prRazred;
```

U set delu se dodeljuje vrednost privatnoj varijabli prRazred.

```
prRazred = value;
```

Promenjiva value je specijalna C# varijabla i predstavlja vrednost koju je korisnik upotrebio prilikom dodele **vrednosti svojstvu**. I ovde se nalazi ključ za proveru vrednosti pre dodele varijabli prRazred.

Pre linije `prRazred = value` treba proveriti da li je sadržaj varijable value ispravan. Ako jeste dodeljuemo tu vrednost varijabli prRazred, u suprotnom najčešće generišemo grešku sa odgovarajućom porukom.

Rezime

Ako želimo da napravimo zaštićena svojstva, kod kojih treba vršiti proveru pre dodele vrednosti procedura je sledeća:

- napraviti privatnu varijablu koja ima isto ime kao željeno svojstvo sa nekim prefiksom. (Ovu konvenciju imenovanja treba shvatiti kao preporuku, a ne kao čvrsto pravilo)
- napraviti javno (public) svojstvo sa željenim imenom i tipom istim kao privatna varijabla
- unutar javnog svojstva uneti get i set delove.
- u get delu korisniku pomoći return naredbe vratiti vrednost privatne varijable
- u set delu proveriti vrednost koju korisnik dodeljuje svojstvu (value) i ako je vrednost validna dodeliti je privatnoj varijabli. U suprotnom prikazati poruku o gresci.

Važno je primetiti da se ovde javno svojstvo koristi kao interfejs ka privatnoj varijabli. Kao neka vrsta zaštite gde se mogu ispitati validne vrednosti pre dodele privatnoj varijabli. Ovo ispitivanje može biti jednostavno, ali i veoma složeno zavisno od potrebe.

Dobro napravljena klasa treba sama sebe da "brani" od pogrešnih vrednosti. Nekada ćete je koristiti vi, a nekada drugi programeri i to stalno treba imati u vidu.

U sledećem nastavku će biti govora o najčešćem načinu za proveru, kako se pokreće i obrađuje greška u slučaju da vrednost nije validna.

Vežbanje:

U klasi Ucenik napravite svojstvo Godiste – godina kada je rođen učenik. Da li je ovom svojstvu potrebna zaštita od pogrešnog unosa i koje su validne vrednosti?

Da li su validne vrednosti fiksne, a ako nisu od čega zavise?

Mala škola programiranja C# (4)

U predhodnom nastavku serijala smo govorili o kreiranju zaštićenih metoda klase. Kreiranjem privatne varijable i parom get i set funkcija smo implementirali interfejs za privatnu varijablu. Na ovaj način smo dobili potpunu kontrolu nad procesom dodele i čitanja vrednosti svojstva.

U ovom nastavku će biti obrađena tehnika kojom se vrednost svojstva može zaštитiti, odnosno ograničiti samo na validne vrednosti.

Čitaoci koji prate ovaj serijal treba da otvore projekt na kome smo radili u prošlom nastavku.

Novi čitaoci treba da urade sledeće:

- pokrenite Visual C# Express Edition
- kliknite na link za kreiranje novog projekta (eng. Create Project)
- kliknite na ikonu "Konzolna Aplikacija" (eng. Console Application)
- napišite sledeći kod (kod koji automatski generiše radno okruženje je sive boje, kod koji treba uneti je plave boje):

```
class Program
{
    static void Main(string[] args) {
        Ucenik cu = new Ucenik();
        cu.Prezime = "Perić";
        cu.Ime = "Petar";
        Console.WriteLine(cu.Prezime);
        Console.WriteLine(cu.Ime);
        Console.ReadKey();
    }
}
```

```
class Ucenik {
    public string Ime;
    public string Prezime;
    private int prRazred;
    public int Razred {
        get {
            return prRazred;
        }
    }
}
```

```
set
{
    prRazred = value;
}
}
```

U klasi postoji javno svojstvo Razred koje definiše u kom razredu je učenik srednje škole. Očigledno, validne vrednosti za ovo svojstvo su brojevi od 1 do 4.

Svojstvo Razred je u stvari interfejs za privatnu varijablu prRazred koja je tip int. Bitno je shvatiti da iako korisnik klase vidi samo svojstvo Razred, za klasu je važna samo vrednost koja se nalazi u privatnoj varijabli prRazred.

Get procedura je zadužena za čitanje svojstva i kao što vidimo u kodu, ona korisniku klase jednostavno vraća vrednost varijable prRazred. S druge strane, za postavljanje vrednosti varijable prRazred je zadužena set funkcija koja vrednost (value) dodeljuje varijabli prRazred.

Da bi vrednost varijable prRazred ograničili na interval od 1 do 4, u set funkciji je potrebno izvršiti provjeru vrednosti varijable value. Vreme je da se upoznamo sa naredbom u C# jeziku koja nam ovo omogućava.

if naredba

Kao i svi ostali programski jezici i C# poseduje ovu osnovnu naredbu za proveru uslova. Sintaksa je jednostavna:

```
if (uslov) {
    // deo koji se izvršava ako je uslov tačan
}
else {
    // deo koji se izvršava ako uslov nije tačan
}
```

Na primer:

```
if (a == 5) {
    Console.WriteLine ("Varijabla a ima vrednost 5");
}
else {
    Console.WriteLine ("Varijabla a ima neku drugu
vrednost");
}
```

U slučaju da varijabla a ima vrednost 5, uslov je tačan i izvršava se blok linija između otvorene i zatvorene vitičaste zagrade odmah ispod if naredbe. U našem slučaju to je samo jedna linija koda, ali ih može biti proizvoljni broj. U suprotnom (eng. else), izvršiće se blok linija ispod naredbe else. Deo else je inače opcioni, ne mora se navoditi ako nam ne treba.

Operator za proveru jednakosti je ==, a osim njega postoje i sledeći:

> veće od	if (a > 5) ...
< manje od	if (a < 5) ...
!= različito	if (a != 5) ...
>= veće ili jednako	if (a >= 5) ...
<= manje ili jednako	if (a <= 5) ...

Uslov može biti i složen kada se sastoji od dva ili više osnovna uslova. U tom slučaju neophodno je navesti i logičke operatore koji vezuju dva ili više uslova. Na primer želimo da proverimo da li varijabla a ima vrednost 5 i varijabla b ima vrednost 3.

Kompletan uslov je tačan samo ako su tačni iskazi a == 5 i b == 3. Logički operand koji na ovaj način spaja dva uslova se zove i (eng. and) i simbolički se obeležava se &&:

if (a == 5 && b == 3)

// izraz je tačan samo ako su oba uslova tačna.

Ako na primer, želimo da proverimo da li je varijabla a > 0 ili varijabla b <10 koristi se logički operand ili (eng. or) koji se simbolički obeležava sa dve vertikalne crte ||:

if (a > 0 || b < 10)

// izraz je tačan ako je tačan ili prvi ili drugi ili oba uslova

Ako se sada vratimo na početni primer gde validna vrednost svojstva Razred, odnosno vrednost privatne varijable prRazred treba da bude u opsegu od 1 do 4 uključujući, ispitivanje bi obavili na sledeći način:

if (value > 0 && value < 5)

U slučaju da je izraz tačan dodelićemo privatnoj varijabli prRazred vrednost value, a u slučaju da nije, potrebno je na neki način obavestiti korisnika klase da je napravio grešku. Dapišite sledeće linije koda:

```
public int Razred {
    get {
        return prRazred;
    }
    set {
        if (value > 0 && value < 5) {
            prRazred = value;
        }
        else {
            // greska
        }
    }
}
```

Na ovaj način smo zaštitili privatnu varijablu prRazred od pogrešnih vrednosti. Dodelujemo joj vrednost samo ako je value veće od 0 i manje od 5, odnosno u intervalu od 1 do 4. Šta raditi ako korisnik klase unese pogrešnu vrednost? Taj slučaj se obrađuje u else delu uslova gde trenutno stoji samo komentar // greska.

Najčešći način je da se generiše greška u izvršavanju koja se prosleđuje korisniku klase. On je dalje odgovoran za obradu ove greške, a ako to ne uradi greška će zauzaviti izvršavanje programa.

Greške i njihovo pokretanje

U .NET okruženju greška je predstavljana klasom Exception. Tačnije rečeno, postoji više vrsta ovakvih klasa koji se vezuju za specifične greške kao na primer rad sa datotekama, rad sa bazama podataka i slično. Klasa Exception predstavlja generalnu klasu za predstavljanje greške koju ćemo mi i koristiti.

Kao i kod svake klase, prvo je neophodno napraviti objekat pomoću naredbe new:

Exception greska = new Exception ("Pogrešna vrednost svojstva Razred");

U ovoj liniji koda smo napravili objekat na osnovu klase Exception i taj objekat je predstavljen varijablom greska. Odmah prilikom kreiranja moguće je, kao ulazni parametar, napisati proizvoljni tekst koji opisuje grešku.

U našem slučaju greška je opisana tekstom "Pogrešna vrednost svojstva Razred" i to je poruka koju će dobiti korisnik klase kada vrednost svojstva Razred postavi van validnog opsega.

Pokretanje ili podizanje greške (eng. raise error) se izvršava naredbom throw gde je argument objekat koji predstavlja grešku:

`throw greska;`

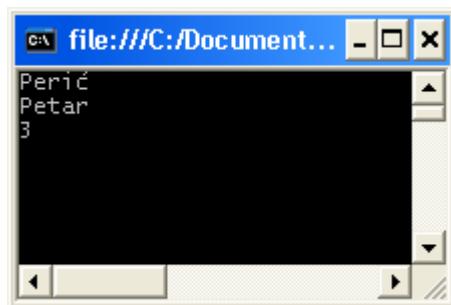
Kada se izvrši throw, klasa prestaje sa radom i ovu grešku prenosi na korisnika klase, koji potom treba da je obradi na odgovarajući način.

Sada treba izmeniti primer kao što je prikazano:

```
static void Main(string[] args)
{
    Ucenik cu = new Ucenik();
    cu.Prezime = "Perić";
    cu.Ime = "Petar";
    cu.Razred = 3;
    Console.WriteLine(cu.Prezime);
    Console.WriteLine(cu.Ime);
    Console.WriteLine(cu.Razred);
    Console.ReadKey();
}
```

i potom izmeniti set deo u klasi:

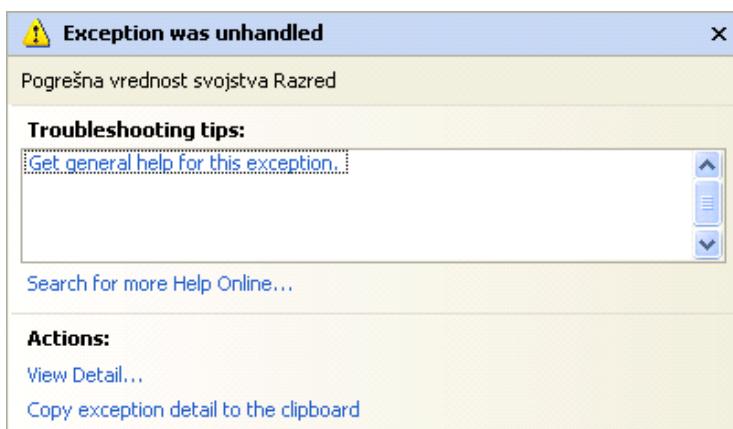
```
set
{
    if (value > 0 && value < 5)
    {
        prRazred = value;
    }
    else
    {
        Exception greska = new Exception
("Pogrešna vrednost svojstva Razred");
        throw greska;
    }
}
```



Ako sada promenimo liniju koja postavlja vrednost svojstva Razred na nevalidnu vrednost, na primer:

`cu.Razred = 7;`

i ponovo pokrenemo program, on se ubrzo zaustavlja i



vraća nas u razvojno okruženje sa našom porukom o grešci:

Obrada grešaka

Da bi sve funkcionalo ispravno, korisnik klase ne treba da dozvoli da bilo koja greška nasilno završi njegovu aplikaciju. Ovo se odnosi na sve moguće greške koje mogu nastati u radu programa, a ne samo na naš primer. Greške možemo podeliti na tri grupe:

1. Sintaksne greške (eng. syntax errors) – rezultat pogrešno napisane naredbe, nedeklarisane varijable i slično. Ove greške najčešće nisu problematične jer program koji ih sadrži ni ne može da se kompajlira.

2. Greške u vreme izvršavanja (eng. runtime errors) – kao u našem primeru, greške koje se mogu ali i ne moraju dogoditi za vreme izvršavanja programa. Najčešće su rezultat neproverenih vrednosti varijabli i pogrešnog unoса korisnika. Relativno se lako pronalaze i ispravljaju.

Zanimljivost:

Opšte prihvaćen termin za grešku u programiranju je "buba" (eng. bug). Otud i termin otklanjanje buba u programu (eng. debugging program).

Odakle ovaj termin?

Prvi kompjuteri, pre pronađaska tranzistora su se zasnivali na vakumskim cevima - kao stari radio aparati i televizori. One su se u toku rada naravno grejale i pri tome emitovale spektar svetlosti veoma privlačan za raznorazne insekte.

Jedna od dužnosti osoba koji su tada održavali računare je bilo redovno čišćenje vakumskih cevi od insekata koji su se na njih zlepili i time im ometali hlađenje. Otud i potiče ovaj termin.

Obrada greške u C# se vrši pomoću naredbi try i catch. U osnovnom obliku sintaksa je sledeća:

```
try
{
    // linije koda koje mogu da generišu grešku
}
catch (Exception varijabla)
{
    // obrada greške
}
```

Upotreba je jednostavna, svaku liniju ili više njih koje mogu generisati grešku treba smestiti u try blok. U slučaju da se generiše greška, tok programa se automatski preusmerava na catch blok.

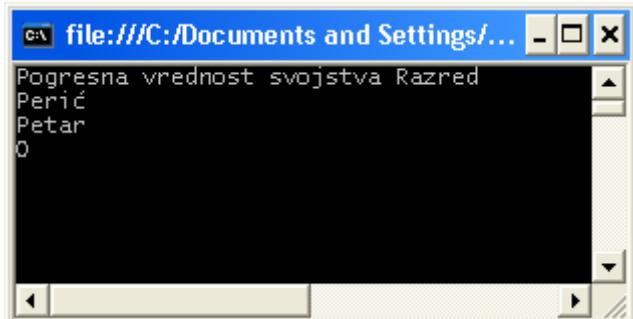
U njemu pišemo kod koji ispravlja grešku ili bar obaveštava korisnika šta nije u redu. Opciono pomoću Exception varijable možemo proveriti koja greška se dogodila, videti njen opis i zavisno od toga izvršiti odgovarajuću akciju.

U našem primeru, greška se može dogoditi na liniji gde se dodeljuje vrednost svojstvu Razred – dakle ta linija se smešta u try blok. U catch bloku koji obrađuje grešku ćemo u konzolnom prozoru ispisati opis greške.

Izmenite kod na sledeći način:

```
static void Main(string[] args)
{
    Ucenik cu = new Ucenik();
    cu.Prezime = "Perić";
    cu.Ime = "Petar";
    try
    {
        cu.Razred = 7;
    }
    catch (Exception ex)
    {
        Console.WriteLine (ex.Message);
    }
    Console.WriteLine(cu.Prezime);
    Console.WriteLine(cu.Ime);
    Console.WriteLine(cu.Razred);
    Console.ReadKey();
}
```

U obradi greške smo u konzoli ispisali tekst greške koristeći Message svojstvo klase Exception. Klasa Exception je u kodu predstavljena objektnom varijablom



ex.

Kada pokrenemo program (obratite pažnju, vrednost svojstva Razred je postavljena na pogrešnu vrednost 7), dobija se sledeći rezultat:

Ovde je važno zapaziti da program nije nasilno zaustavljen zbog greške, već je ona obrađena i program je nastavio sa radom.

Takođe vrednost svojstva Razred je ostala nepromenjena, odnosno ostavljena na inicijalnu vrednost nula.

U sledećem nastavku serijala počinjemo sa programiranjem Windows formi – odnosno sa aplikacijama koje imaju korisnički interfejs. Konzolne aplikacije su bile neophodni uvod u objektno programiranje, varijable i osnovne programske strukture.

Mala škola programiranja C# (5)

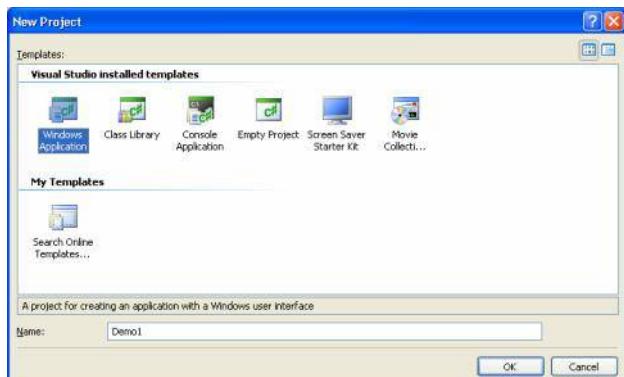
Kreiranje Windows aplikacija

U ovom i sledećim nastavcima ćemo se posvetiti kreiranju Windows aplikacija u radnom okruženju Microsoft Visual C# Express edition. Na početku, kada su se pojavile prve verzije Windowsa, programiranje Windows aplikacija je bilo veoma teško i dostupno samo odabranim profesionalcima. Pojavom vizualnih alata i takozvanog RAD (Rapid Application Development – rapidni razvoj aplikacija) sistema, kreiranje ovakvih aplikacija je značajno pojednostavljeni i postalo je dostupno širokom krugu programera. Kako je ovo postignuto? U predhodnim poglavljima smo učili objektno programiranje i u njemu je ključ. Svi elementi koji se koriste u programiranju Windows aplikacija su "prefabrikovani" - stoje nam na raspolaganju sa svojom paletom svojstava, metoda i događaja u vidu gotovih klasa. Nama samo ostaje da ih organizujemo na način koji nam je potreban, postavimo im odgovarajuća svojstva i napišemo kod kao odgovor na akcije korisnika – događaji klasa.

Postavljamo se u ulogu arhitekte koji ima na raspolaganju veliki broj elemenata koji čine jednu kuću. Na nama je da dobro upoznamo te elemente, ukomponujemo ih i podesimo da harmonično funkcionišu.

Na vrhu hijerarhije elemenata je sama aplikacija, odnosno objekt Application. Ispod njega se nalazi bar jedna, a obično i više Windows forme. Svaka od njih sadrži standardne elemente korisničkog interfejsa Windowsa koje nazivamo kontrole. Svakom korisniku Windowsa su poznati ovi elementi u koje spadaju komandna dugmad, liste, padajuće liste, izbor opcija, meniji i tako dalje.

U ovom i sledećim nastavcima ćemo se upoznati sa funkcionalnošću koje nude ove kontrole i naučiti da ih programiramo.

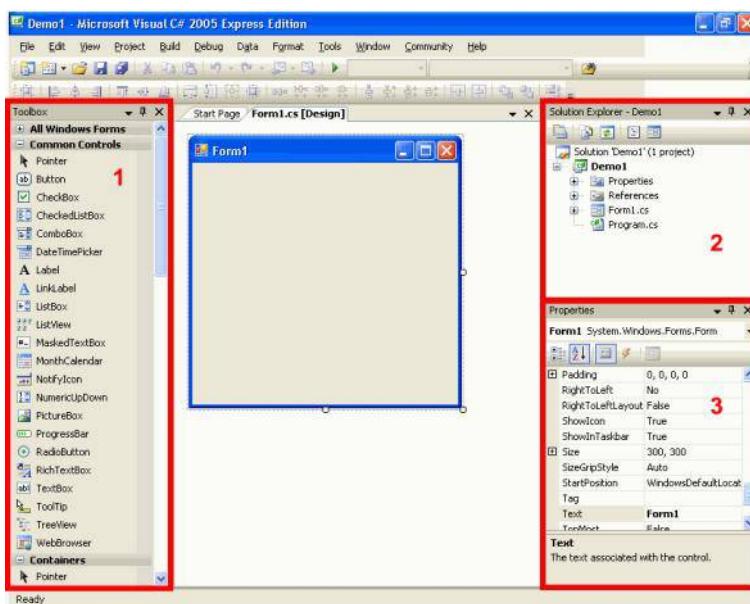


Posle par sekundi, nov projekt je kreiran i u njemu se inicijalno nalazi početna forma koja je prikazana na ekranu. U ovom momentu možete čak i pokrenuti program (F5 funkcionalni taster) i videti praznu formu na ekranu. Trenutno naša aplikacija ne izgleda ništa posebno, jednostavna prazna forma – ali ako malo bolje pogledamo ona ima sve funkcionalnosti Windows forme: naslov, ikonu, mogućnost da je povećamo preko celog ekrana ili minimizujemo i konačno zatvorimo, što ujedno i zaustavlja izvršavanje projekta.

Odakle joj sve ove funkcionalnosti kada još uvek nismo ništa programirali? U predhodnim nastavcima ovog serijala bilo je govora o klasama, a naša forma je naslednik postojeće klase Form koja se nalazi u .NET Frameworku. Svi naslednici bilo koje klase nasleđuju njene funkcionalnosti uz mogućnost dodavanja novih. Dakle puno stvari za nas je unapred već urađeno.

Kada radimo Windows aplikacije razlikujemo dva osnovna dela:

- Dizajn korisničkog interfejsa u kome "crtamo" i raspoređujemo kontrole na formi. Ovde takođe vršimo i podešavanje svojstava i ponašanje kontrola.
- Kodiranje, deo u kome pišemo kod kao odgovor na akcije korisnika, na primer klik mišem na komandno dugme, izbor iz liste, zatvaranje forme i slično.



Na narednoj slici je dat prikaz okruženja i njegovih elemenata potrebnih za razvoj Windows aplikacija:

1. Paleta sa alatima (Eng. Toolbox) – lista kontrola koje se mogu koristiti na formi
2. Pretraživač rešenja (Eng. Solution Explorer) – prikaz svih projekata i njihovih elemenata (forme, klase, resursi i ostalo)
3. Svojstva (Eng. Properties) – prikazuje svojstva izabrane kontrole ili forme

Napomena:

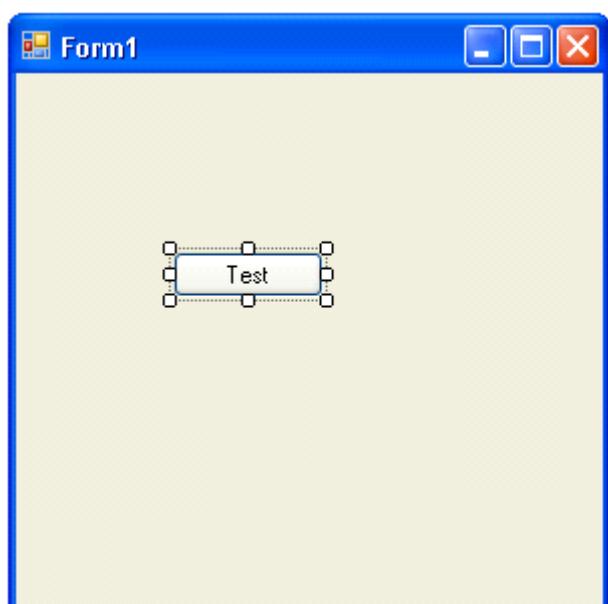
Ako ne vidite prozor svojstava iz glavnog menija izaberite stavku View i kliknite na Properties Window.

Prva aplikacija

Sada ćemo napraviti jednostavnu aplikaciju sa samo jednim komandnim dugmetom. Kada kliknemo na njega ispisaćemo poruku "Moja prva Windows aplikacija".

Na postojeću formu treba postaviti jedno komandno dugme. Svaka kontrola se postavlja tako što se na nju klikne i potom prevuče na formu. Dakle kliknite na stavku Button (paleta sa alatima) i prevucite je na formu. Kada je kontrola na formi, možemo joj naknadno menjati poziciju i dimenzije po želji. Ako nam ne treba, jednostavno na nju kliknite mišem i obrišete je pomoću tastera Delete na tastaturi.

Na komandnom dugmetu inicijalno piše "button1" što ćemo izmeniti u "Test". Kliknite mišem na dugme i u prozoru svojstva (Eng. Properties) nađite svojstvo pod nazivom "Text". Vidimo da tamo piše "button1". Izmenite ga u "Test" i pritisnite Enter taster. Sada naša forma sa dugmetom treba da izgleda kao što je prikazano na slici.



Kada želimo da napišemo kod koji se izvršava kada korisnik nešto uradi sa dugmetom postoji više načina. Najjednostavniji je da uradite dupli klik na komandno dugme.

Sada je otvoren prozor za pisanje koda i kurzor se nalazi tačno na poziciji gde treba da počnemo pisanje.

```
private void button1_Click(object sender, EventArgs e)
{
}
```

Obratite pažnju na naziv funkcije button1_Click koji ukazuje da se radi o kontroli koja se zove button1 i da obrađujemo događaj koji se izvršava kada korisnik klikne na njega.

Možda je na prvi pogled zbumujuće što je ostalo button1, a ne Test – ali mi smo izmenili natpis na dugmetu, ne i njegovo ime koje je ostalo button1.

Ovo prikazuje još jednu važnu činjenicu: svaka kontrola na formi ima svoje *jedinstveno ime* pomoću koga se vezuju događaji, ali takođe svojstva i metode kontrole.

Svojstvo kontrole koje određuje njeno ime se naziva Name i obavezno ga je definisati. Prilikom postavljanja dugmeta na formu, Visual Studio mu je automatski dodelio ime "button1". Ovo znači da bi sledeće dugme dobilo ime "button2" i tako redom. Kasnije ćemo govoriti o nazivima kontrola, za sada ćemo ostaviti da ih postavlja Visual Studio.

U okviru vitičastih zagrada napišite sledeću liniju koda:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Moja prva Windows aplikacija");
}
```

MessageBox predstavlja standardni dijalog za poruke u Windowsu. Metoda Show prikazuje ovaj dijalog sa ulaznim parametrom koji predstavlja tekst koji treba prikazati.

Kao i ranije vodite računa o velikim i malim slovima u kodu jer ih C# programski jezik razlikuje.

Pokrenite program (F5 funkcijski taster) i kliknite na dugme.

Ako ste sve ispravno napisali, dobićete poruku kao što je prikazano na sledećoj slici:



Kliknite na dugme "OK" i potom zatvorite formu kako bi zaustavili program.

U razvojnom okruženju obratite pažnju na dve kartice (Eng. Tab) iznad prozora za pisanje koda: Form1.cs i Form1.cs [Design].

Klikom na njih prebacujete se u prikaz koda (Form1.cs) i dizajniranja forme (Form1.cs [Design]). Ovu proceduru ćemo često raditi prilikom izrade aplikacije.

Prvo postavljamo i uređujemo kontrole na formi i potom pišemo kod za njihove događaje. Pokrenemo program da bi proverili da li sve funkcioniše i tako ciklično za svaku formu u projektu.

U sledećem delu serijala nastavljamo sa kontrolama i naučićemo kako se dodaju i koriste nove forme u projektu.

Mala škola programiranja C# (6)

Rad sa formama

Svaka Windows aplikacija se sastoji najmanje od jedne forme (formulara), međutim najčešće ih ima nekoliko, pa čak i nekoliko desetina u složenijim aplikacijama. Jasno je da treba naučiti načine kako programski otvoriti i zatvoriti željenu formu, kao i načine postavljanja osobina forme.

Ovo nas ponovo vraća na objektno programiranje, jednostavno zbog toga što svaka forma prilikom kreiranja aplikacije predstavlja jednu klasu. Koncept je jednostavan: obično se počinje od kreiranje "glavne" forme – ona koja prva otvara prilikom pokretanja same aplikacije i onda se dodaju sve ostale koje su nam potrebne.

Na primer, prilikom pokretanja programa MS Word, glavna forma je zapravo deo Worda gde unosimo i uređujemo tekst. Svi ostali dijalozi kao što je ekran za podešava-

Napomena:

Pažljivi čitaoci i posmatrači će primetiti da se prilikom pokretanja Worda zapravo prvo pokreće ekran sa Word logotipom i raznim informacijama. Ovaj ekran se posle par sekundi automatski yatvara (zavisno od brzine računara) i tek onda se prikazuje prozor za unos teksta. Ovo važi za sve office aplikacije i skoro sve ostale programe na tržištu.

Početni prozori ovog tipa se zovu "Splash screen" po uzoru na "Splash news" i označavaju prikaz očigledne informacije koju je nemoguće prevideti. Osim manje-više značajnih informacija koje su prikazane na ovom ekranu, oni imaju još jednu, bitniju namenu: dok gledate ovaj ekran, istovremeno, u pozadini se vrši inicijalizacija aplikacije koju ste pokrenuli.

Psihološki efekat je da čim korisnik pokrene aplikaciju odmah vidi neku aktivnost na ekranu - ne treba da čeka par ili više sekundi i da se pita da li je sve u redu.

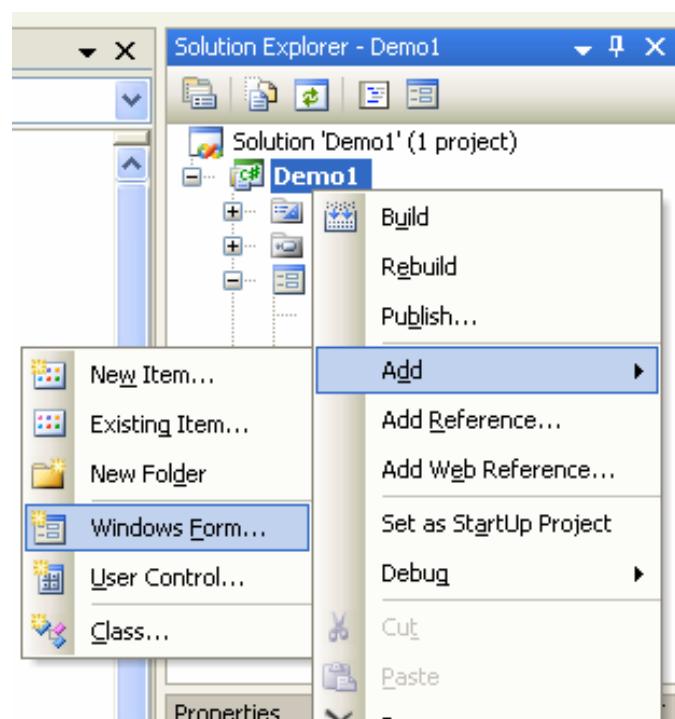
nje opcija, umetanje tabela i mnogi drugi, predstavljaju zasebne forme koje se otvaraju kao odgovor na akciju korisnika.

Teoretski gledano, svaka forma predstavlja klasu koja je nasleđena od bazne .NET Framework klase Form iz imenovanog prostora System.Windows.Forms. Bazna klasa pruža mnoge fukcionalnosti našoj klasi (formi) koje može mo odmah koristiti.

Prilikom izvođenja programa, moramo programski instancirati ovu klasu kako bi bilo šta dalje mogli raditi. U vreme dizajniranja forme, možemo po potrebi postavljati različite osobine – svojstva forme, ali ih takođe i naknadno menjati u vreme izvršavanja.

Dodavanje nove forme u program

Otvorite primer koji smo radili u predhodnom broju časopisa (Demo1), ili možete kreirati nov projekt: pokrenite Visual C# Express Edition, iz opcije File izaberite stavku New Project, u otvorenom dijalogu izaberite Windows Application. Za ime projekta ukucajte Demo1 (linija Name:). Na kraju kliknite na dugme OK. U oba slučaja se dobija kreiran projekt pod nazivom Demo1 sa prvom i za sada jedinom početnom formom.



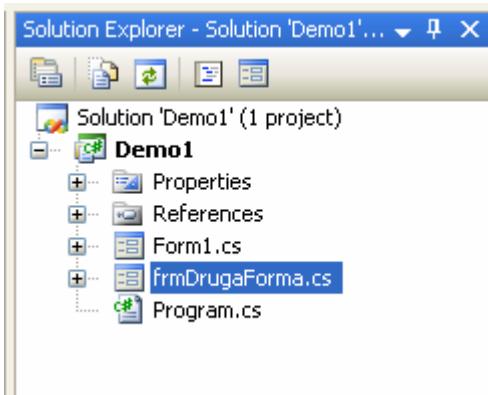
Sada u Project Exploreru uradite desni klik na projekt Demo1, iz padajućeg menija izaberite Add i iz podmenija izaberite Windows Form kao što je prikazano na slici:

Na ovaj način se otvara opšti dijalog za dodavanje nove stavke u projekat, sa izabranom ikonom Windows Form. U ovom dijalogu možemo odmah dati i ime novoj formi (klasi) što ćemo i učiniti.

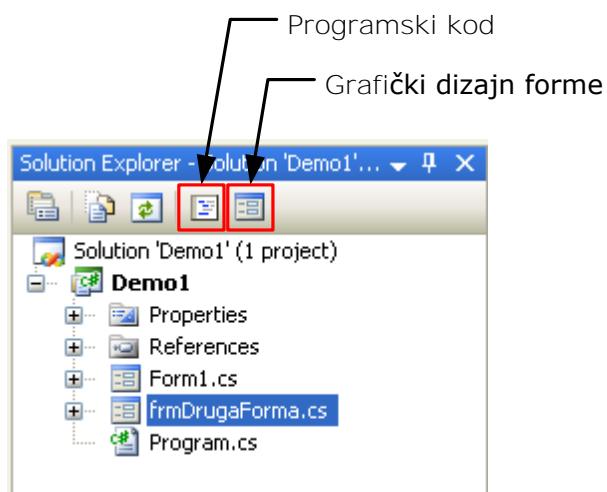
Ranije sam pomenuo da tehnički gledano nije neophodno, ali je svakako korisno i predstavlja dobru praksu davati unificirana imena svim klasama i objektima.

U ovom smislu ćemo novoj formi dati naziv frmDrugForma (jedna reč, bez razmaka) koji treba upisati u liniju Name i na kraju kliknuti na dugme Add.

Dodata forma je prikazana na ekranu, a u Solution Exploreru je takođe prikazana sa specifičnom ikonom koja označava sve forme u projektu:



Iz Solution Explorera po potrebi možemo prilikom projektovanja i pisanja koda otvarati potrebnu formu i to u dva režima: režim za grafičko dizajniranje forme i režim za pisanje programskog koda koji se nalazi na formi.



Potrebno je izabrati formu i potom kliknuti na željeno dugme kao što je prikazano na predhodnoj slici. Opciono možete uraditi i dupli klik na naziv forme, čime se ona uvek otvara u grafičkom režimu.

Sada želimo da na početnoj formi (Form1) postavimo standardno komando dugme i napišemo kod koji će otvoriti drugu formu (frmDrugForma) kada korisnik klikne na ovo dugme.

Pošto sada radimo na klasi Form1 i to u dizajn režimu, u Solution Exploreru uradite dupli klik na klasu Form1.

Na formu je dalje potrebno postaviti komandno dugme. Kao i u prošlom nastavku serijala iz palete sa alatima (Toolbox) izaberite komandno dugme (Button) i nacrtajte ga na površini forme. Alternativno možete uraditi i dupli klik mišem na Button stavku iz palete sa alatima.

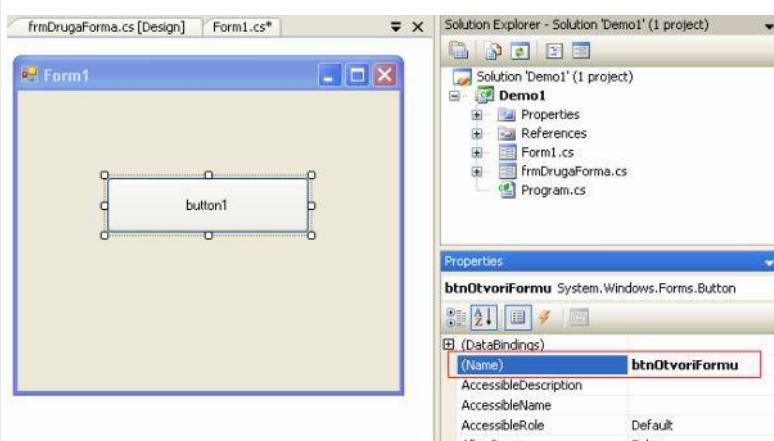
Svojstvo Name

Dugme je automatski dobilo ime "Button1", a na njemu se nalazi istovetni tekst "Button1". Sada je vreme da se upoznamo sa važnim svojstvom svakog elementa u kreiranju korisničkog interfejsa. Svojstvo Name predstavlja jedinstveno ime elementa na koje se referišemo kada postavljamo sva ostala njegova svojstva, pišemo kod za njegove događaje i koristimo njegove metode. Svojstvo Name zapravo predstavlja ime klase koja predstavlja taj element, na potpuno isti način kao što se u programiranju imenuju promenjive.

- Pravila za zadavanje imena su jednostavna:
- maksimalna dužina za ime je 128 karaktera
 - ime može sadržati slova i brojeve, ali ne može početi brojem
 - ime ne sme sadržavati specijalne karaktere i razmake

Poštujући ova pravila i preporučen prefiks, Name svojstvo dugmeta ćemo postaviti na btnOtvoriFormu.

Kliknite jedanput na dugme i u prozoru Properties

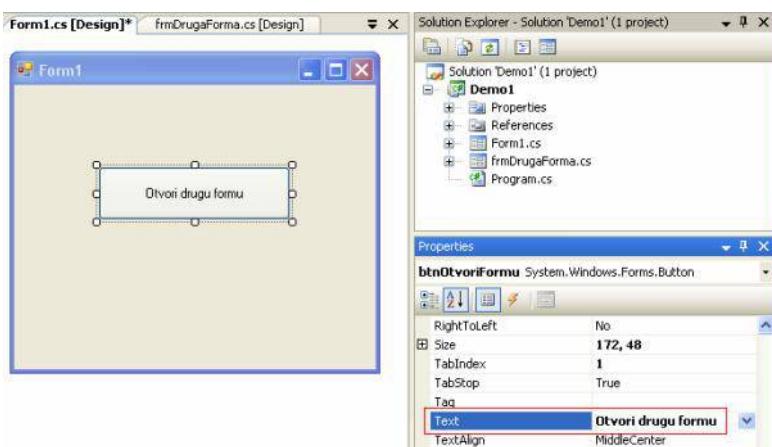


(dole desno u razvojnog okruženju) pronađite svojstvo Name i promenite ga u btnOtvoriFormu kao što je prikazano na slici i pritisnite Enter taster.

Primetićete da se natpis na dugmetu nije promenio. Očigledno natpis na dugmetu ne prikazuje njegovo ime već nešto drugo.

Svaki element korisničkog interfejsa koji ima neki natpis na sebi, uključujući i formu ima svojstvo Text koje određuje ovaj natpis.

U listi svojstava (Properties dijalog) za komandno dugme pronađite Text svojstvo i izmenite ga na "Otvori drugu formu". Text svojstvo ne podleže pravilima za zadavanje imena klasama i varijablama, tako da možete upisati šta god želite. Na kraju forma za dugmetom treba da izgleda ovako:



Kodiranje

Na kraju ostaje da upišemo kod koji će otvoriti drugu formu. Duplim klikom na dugme otvaramo kod za pisanje događaja. Dugme poznaje puno događaja, međutim ubedljivo najčešće se koristi Click događaj, što nam razvojno okruženje inicijalno i nudi.

Pošto forma frmDrugForma predstavlja klasu, pre korišćenja moramo napraviti instancu te klase.

Deklariraćemo varijablu koja će biti odgovarajućeg tipa i inicijalizovati je kao pokazivač na instancu klase frmDrugForma. Ime varijable nije bitno, a mi ćemo koristiti ime MojaForma.

Na mestu gde se nalazi kurzor ukucajte sledeće dve linije koda:

```
frmDrugForma MojaForma = new frmDrugForma();
MojaForma.Show();
```

U prvoj liniji deklarišemo varijablu MojaForma koja je tipa frmDrugForma i odmah joj dodeljujemo novu instancu (ključna reč new) objekta koji je baziran na klasi frmDrugForma.

Klasa (pa i objekat) tipa Form ima metodu kojom se forma iscrtava i prikazuje na ekranu. Metoda Show() je iskorišćena u drugoj liniji. Sintaksa poziva metode bilo kog objekta je jednostavna:

ImeObjekta.NazivMetode (argumenti)

Verzija metode Show koju mi koristimo nema argumente, ali svejedno moramo napisati otvorenu i zatvorenu zgradu.

Pokrenite program (F5 funkcionalni taster) i proverite kako radi.

Modalne i nemodalne forme

Ako malo eksperimentišete primetićete sledeće: kada kliknete na dugme otvara se druga forma preko prve, ali uvek možete kliknuti mišem na prvu formu čime ona sada prekriva drugu. Ovakva relacija između dve forme se zove "nemodalne forme".

Često je u praksi potrebno drugačije ponašanje. Nije dozvoljeno ovakvo preklapanje formi, već korisnik mora zatvoriti drugu formu da bi se vratio na predhodnu. Ovakva relacija između formi se zove "modalne forme" (često je u upotrebi i termin "Dialog forma"). Kao primer modalnog ponašanja možete opet pogledati skoro sve dijaloge u MS Wordu.

Da bi implementirali modalnu relaciju između naše dve forme, potrebno je umesto metode Show() koristiti metodu ShowDialog(), takođe bez argumenta.

Zaustavite program ako je pokrenut, i izmeniti drugu liniju koda:

[MojaForma.ShowDialog\(\);](#)

Ponovo pokrenite program i sada vidite da se ne možete prebaciti na prvi formu dok ne zatvorite drugu. Modalni (dialog) način otvaranja forme se u praksi znatno češće koristi.

Zatvaranje forme

Otvorenou formu možemo uvek zatvoriti na standardni način klikom na dugme x u njenom gornjem desnom uglu. Međutim često je potrebno programski zatvoriti formu. Da bi ovo uradili moramo se predhodno upoznati sa pokazivačem this.

this je varijabla koja pokazuje na objekat u kome se nalazi – u našem slučaju na formu. Kada bilo gde u kodu koji se nalazi u formi napišete this vi se zapravo referišete na sam form objekat te su dostupne sve metode i svojstva ne samo forme, već i svih elemenata koje se nalaze na njoj.

Napomena:

Za programere u Visual Basicu i u VB.NET postoji istovetni pokazivač koji se zove Me.

Metoda koja zatvara formu je Close() bez argumenta. Tako da kada bilo gde u kodu na formi napišemo:

```
this.Close();
```

ovo znači bezuslovno zatvaranje forme u kojoj se ovaj kod nalazi.

Vežba

Za vežbu na kraju ovog nastavka pokušajte da uradite sledeće:

Na drugoj formi (frmDrugaForma) postavite dugme, dajte mu naziv btnZatvoriFormu i neka na njemu bude tekst "Zatvori formu".

Kada korisnik klikne na ovo dugme treba naravno zatvoriti ovu formu.

Mala škola programiranja C# (7)

Windows kontrole

Osnove svake windows aplikacije predstavljaju formulari (eng. forms) koji se po potrebi otvaraju i zatvaraju, zavisno od toka i logike korisničkog okruženja. Na svakom formularu se nalazi više objekata koji predstavljaju korisnički interfejs – windows kontrole. U narednim poglavljima ćemo se upoznati sa osnovnim windows kontrolama, njihovom namenom, svojstvima, metodima i događajima.

Osnovno za svaku kontrolu je svojstvo Name koje predstavlja njen jedinstveni identifikator u okviru formulara na kome se nalazi. Ovo znači da na jednom formularu ne možemo imati dve ili više kontrola koje se isto zovu.

Međutim, na dva ili više formulara to je moguće. Svojstvo Name treba zadavati po istom principu po kome zadajemo imena varijabli u programskom kodu - bez razmaka i specijalnih karaktera, možemo imati brojeve u nazivu, ali naziv mora početi slovom. Kao što ćemo videti kasnije, svojstvo name zapravo i jeste objektna varijabla koja je pokazivač na instancu klase odgovarajuće kontrole.

Dobra praksa je davati prefiks imenu svake kontrole zavisno od njenog tipa. Nije obavezno, ali kasnije značajno olakšava preglednost programskog koda. Lista preporučenih prefiksa za svaku kontrolu se nalazi na lokaciji

<http://msdn2.microsoft.com/en-us/library/ms229045.aspx>

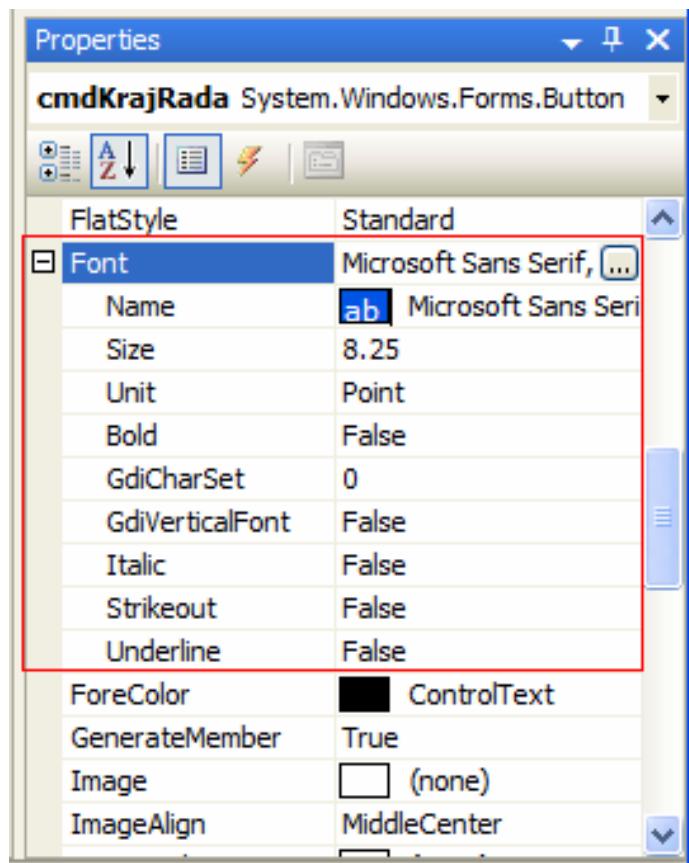
Komandno dugme (eng. command button)

Komandno dugme (ili samo dugme), predstavlja najčešći i verovatno najjednostavniji kontrolu. Omogućava korisniku da klikom na njega pokrene odgovarajuću akciju – segment programskog koda. Ključno svojstvo dugmeta je svojstvo Text kojim se postavlja natpis na njemu. Generalno, svaka kontrola koja na sebi poseduje neki natpis ima svojstvo Text.

Ovo svojstvo ima još jednu mogućnost: ako postavite znak & ispred bilo kog karaktera u Text svojstvu, taj karakter je podvučen i predstavlja skraćenicu sa tastature za klik akciju dugmeta.

Na primer, kada svojstvo Text dugmeta postavimo na **Lista &učenika**, prikazuje se Lista učenika a skraćenica sa tastature je Alt+U. Jasno, na jednoj formi je potrebno imati jedinstvene skraćenice kako bi sve funkcionalo kako treba.

Za dugme, kao i za većinu ostalih kontrola možemo postaviti font koji se koristi kao i njegove atribute (podebljan, ukošen, podvučen). Svojstvo Font nam omogućava sve ovo i na sledećoj slici je prikazano sa svim svojim



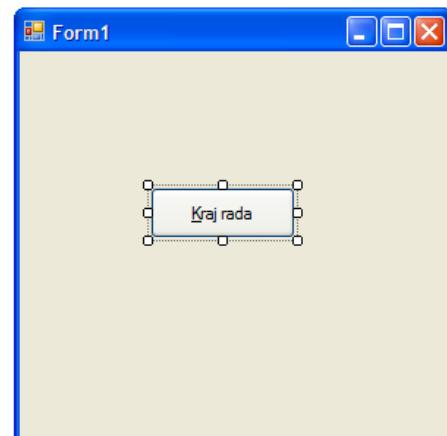
atributima.

Od događaja, najčešće se koristi Click, koji se pokreće kada korisnik klikne na dugme.

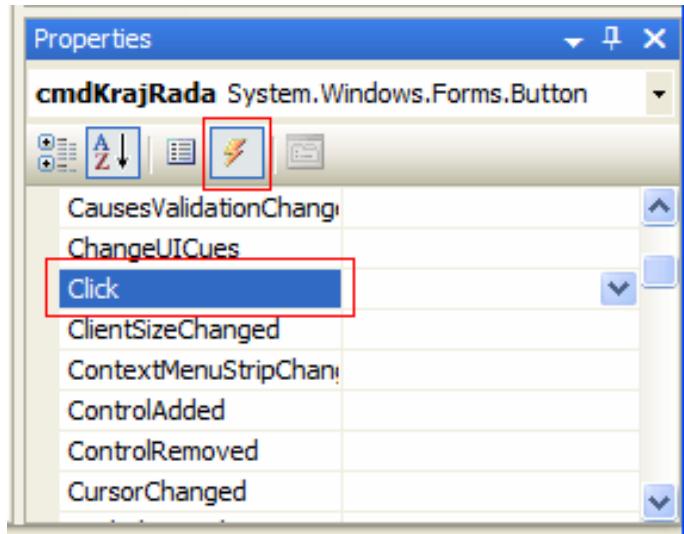
Vežba:

Pokrenite nov projekt i kreirajte novu Windows aplikaciju.

Postavite komandno dugme na početni formular. Svojstvo Name postavite na cmdKrajRada, a svojstvo Text na &Kraj rada.



Kod za različite događaje kontrole (eng. control events) pišemo na sledeći način: mišem izaberite dugme i kliknite na ikonu (narandžasta munja) u dijalogu svojstava koja daje listu svih raspoloživih događaja izabranog elementa, kao što je prikazano na slici.



Pronadite Click događaj i mišem uradite dvostruki klik na njega. Na ovaj način ste otvorili prozor za pisanje programskog koda koji se izvršava za izabranoj akciji – Click u našem slučaju.

U ovoj vežbi, želimo da se program završi kada korisnik klikne na dugme.

Na mestu kurzora upišite sledeću liniju koda:

```
Application.Exit();
```

Koristimo Application objekat koji predstavlja pokazivač na aplikaciju i njegov metod Exit koji bezuslovno završava rad aplikacije i oslobađa sve resurse koje ona koristi.

Pokrenite aplikaciju (F5 funkcijski taster) i kliknite na dugme. Uočite da isto možemo postići skraćenicom Alt+K, jer smo je postavili pomoću Text svojstva dugmeta.

Što se tiče velikog broja ostalih svojstava dugmeta, probajte sami da eksperimentišete sa njima.

Većina njih je orientisana na grafički izgled dugmeta, pa posebno obratite pažnju na svojstvo Image i njemu slična.

Labela (eng. Label)

Labela daje mogućnost prikaza teksta na formularu koji može biti samostalan, ali se obično vezuje za neku drugu kontrolu i na taj način opisuje njenu značenje.

Najčešće se postavlja uz TextBox kontrolu koja je opisana sledeća. Ključno svojstvo kontrole je Text koje takođe poseduje mogućnost dodele skraćenice sa tastature kao što smo radili na dugmetu.

Linija za unos teksta (eng. TextBox)

Kontrola koja omogućava korisniku da unese podatke koji se dalje koriste u aplikaciji. Relativno složena kontrola koja se veoma često koristi prilikom građenja korisničkog okruženja. Omogućava unos teksta u jednoj ili više linija. Možemo ograničiti mogući broj unetih karaktera, postaviti font koji se koristi i odrediti druge grafičke elemente.

Ključno svojstvo ove kontrole je takođe svojstvo Text koje se može postavljati i čitati za vreme izvršavanja aplikacije i na taj način se vrši obrada unosa korisnika.

Svojstvo Multiline označava da li je moguća jedna (False) ili više linija teksta (True).

Ako je svojstvo Multiline postavljeno na True, obično se postavlja i svojstvo ScrollBars koje ovakvom višelinjskom tekstu dodaje mogućnost pomeranja po vertikali i horizontali.

Svojstvo MaxLength definije maksimalni broj karaktera koji korisnik može uneti. Inicijalno je postavljen na 32767, što je u praksi obično previše – uvek će biti znatno manje.

Od događaja se najčešće koristi TextChanged. Ovaj događaj se automatski pokreće svaki put kada korisnik unese ili obriše karakter u kontroli. Može se iskoristiti za proveru unetog sadržaja, on line pretragu i tome slično.

Vežba:

Želimo da napravimo jednostavan forumular u kome korisnik može da unese svoje ime i prezime.

Na formular postavite dve Label i dve TextBox kontrole i to sledećim redom:

1. Postavite prvu Label kontrolu i postavite joj sledeća svojstva:

Name: lblIme i Text: &Ime

2. Postavite prvu TextBox kontrolu sa svojstvom

Name: txtIme i MaxLength: 15

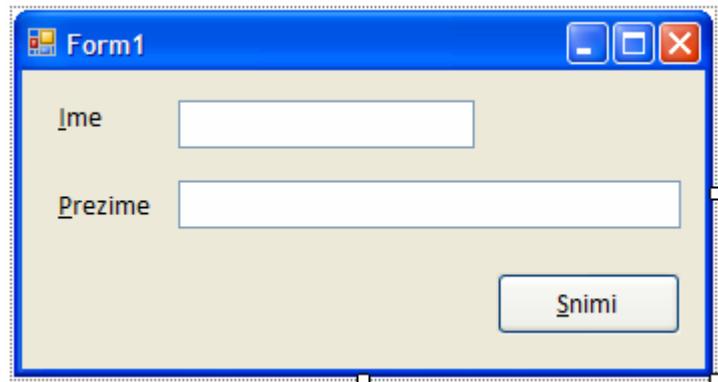
3. Postavite drugu Label kontrolu sa svojstvima:

Name: lblPrezime i Text: &Prezime

4. Postavite drugu TextBox kontrolu sa svojstvima

Name: txtPrezime i MaxLength: 25

5. Na kraju postavite dugme sa svojstvima



Name: cmdSnimi i Text: &Snimi

Formular treba da izgleda ovako:

U realnosti klikom na dugme cmdSnimi bi unete podatke snimili u bazu podataka, ali za sada ćemo ih samo ispisati u dijalogu za poruke.

Kao što je opisano na početku ovog nastavka, otvorite kod za Click događaj komandog dugmeta i napišite sledeće dve linije koda:

```
MessageBox.Show (txtIme.Text);
MessageBox.Show (txtPrezime.Text);
```

Objekat MessageBox predstavlja dijalog za poruke. Poseduje metodu Show koja ima jedan parametar – tekst koji treba prikazati. U ovom primeru prikazujemo sadržaj (uneti tekst) u kontrolama txtIme i txtPrezime.

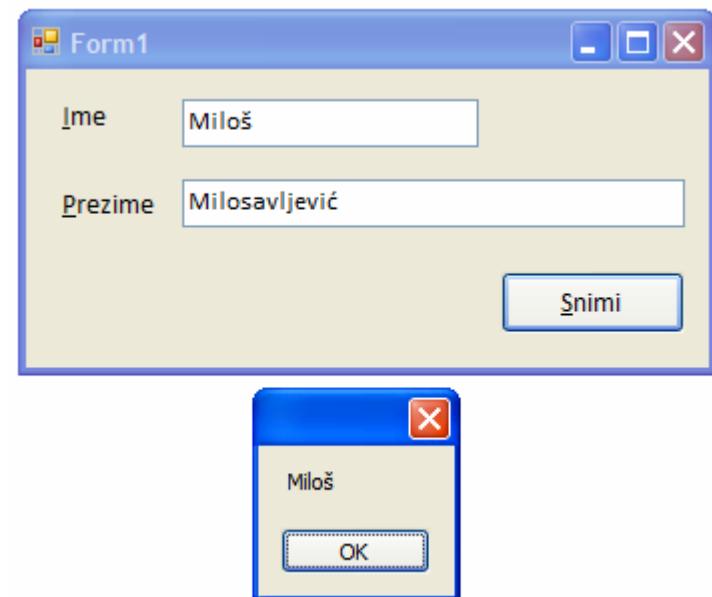
Vidimo i sintaksu pristupa svojstvu bilo koje kontrole: ImeKontrole.SvojstvoKontrole.

Na ovaj način možemo čitati ali i postavljati vrednosti svojstava kontrola.

Napomena:

Postoji više varijanti metode Show MessageBox

Pokrenite primer, unesite svoje ime i prezime i kliknite na dugme Snimi. Ako ste sve upisali kako treba, dobijete sledeći rezultat sa dva uzastopna dijaloga za poruke.



Analiza primera:

Pošto smo postavili skraćenice na obe Label kontrole, ispitajte kako funkcionišu. Kombinacija Alt+I treba da kurzor postavi na TextBox za unos imena, a Alt+P na TextBox za unos prezimena. Alt+S će pokrenuti Click događaj dugmeta. Kako je ovo postignuto?

U svakoj Windows formi postoji pojma TabOrder. On definiše redosled kojim se vrši navigacija kada korisnik klikne na Tab ili Shift+Tab taster na tastaturi.

Veoma je bitno postaviti logični redosled kretanja po formularu. Svaka kontrola poseduje numeričko svojstvo TabIndex koje počinje od broja 0 pa na dalje.

U našem primeru prva labela imaTabIndex svojstvo postavljeno na 0, prvi TextBox na 1, i tako redom.

Drugi pojam koji treba znati je Focus.

On predstavlja situaciju kada kontrola postaje aktivna. Na primer kada je cursor postavljen u kontroli za unos prezimena, kažemo da je TextBox txtPrezime dobio fokus.

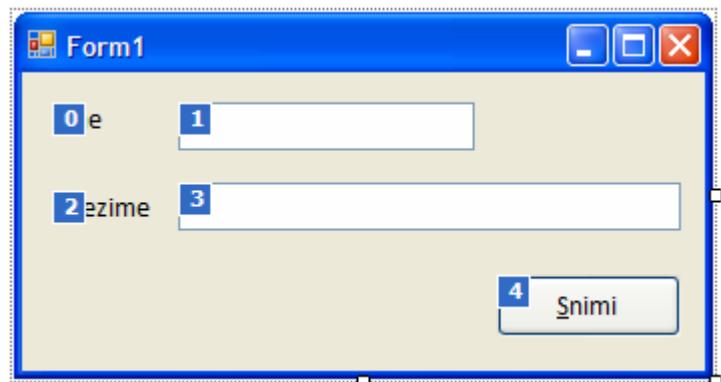
Ne mogu sve kontrole da imaju fokus. Očigledno, Label kontrola ne može da dobije fokus.

Šta se događa kada korisnik na tastaturi klikne Alt+P?

Labela lblPrezime ne može da dobije fokus, te ga automatski prenosi na prvu sledeću kontrolu koja to može – u našem primeru to je TextBox txtPrezime.

Prva sledeća kontrola se računa upravo po svojstvu TabIndex.

Svaka kontrola koju postavimo na formular automatski dobija svoj TabIndex u skladu sa redosledom postavljanja. Često se međutim u praksi postavlja potreba za ručnom izmenom redosleda kontrola. To možemo uraditi tako što ćemo svakoj kontroli pojedinačno postavitiTabIndex svojstvo, što može biti nekomfornto kada imamo puno kontrola na formi.



Drugi način je zgodniji i manje sklon greškama.

Zaustavite program i kliknite na formular. Iz menija View izaberite stavku Tab Order.

Na ovaj način je prikazanoTabIndex svojstvo svake kontrole na formularu.

Nama ostaje da sa po jednim klikom miša na svaku kontrolu, redosledom koji je potreban postavimo odgovarajućiTabIndex. Probajte ovo da uradite za svaku kontrolu na našoj formi, i kada završite ponovo u meniju View isključite opciju Tab Order.

U sledećem nastavku serijala nastavljamo sa drugim, složenijim Windows kontrolama.

Mala škola programiranja C# (8)

Windows kontrole

Prilikom kreiranja korisničkog interfejsa u Windowsu, na raspolaganju nam je veliki broj grafičkih elemenata – kontrola. U ovom nastavku se upoznajemo sa malo složenijim kontrolama koje nam nude specifične funkcionalnosti.

CheckBox kontrola

Ova kontrola omogućava korisniku da markira ili demarkira neku opciju. Ako imamo više CheckBox kontrola na formularu, one su međusobno nezavisne – mogu se proizvoljno markirati ili demarkirati.

Ključno svojstvo ove kontrole je svojstvo Checked koje može imati dve vrednosti - True (da) i False (ne). Svojstvo Checked možemo postavljati i čitati kako u režimu dizajniranja tako i u režimu izvršavanja aplikacije.

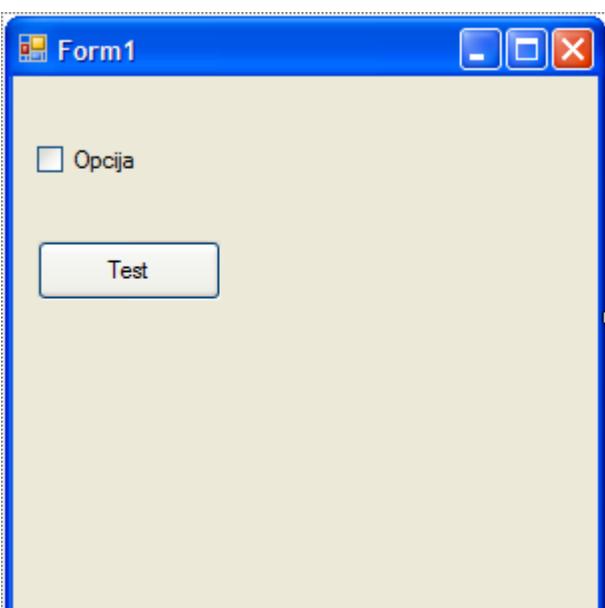
U kodu se najčešće koristi kroz IF uslov koji vrši ispitivanje da li je CheckBox markiran ili ne i u zavisnosti od toga se vrši dalje grananje koda.

Opis značenja CheckBox kontrole se definiše kroz njeno Text svojstvo. Nije potrebno postavljati zasebnu Label kontrolu – ona je integrisana u CheckBox.

Vežba:

Pokrenite C# Express razvojni alat, kreirajte novu Windows aplikaciju i na inicijalnu formu postavite jednu CheckBox i jednu Button kontrolu.

Za Text svojstvo CheckBox kontrole unesite "Opcija" (bez znakova navoda), a za Text svojstvo Button kontrole unesite "Test". Formular treba da izgleda kao što je prikazano na slici:



Klikom na dugme želimo da proverimo stanje CheckBox kontrole i da zavisno od toga ispišemo odgovarajuću poruku.

Da bi ovo uradili, moramo se upoznati sa naredbom za ispitivanje uslova u C# programskom jeziku. Poznavaoci C ili C++ programskega jezika će videti da je sintaksa identična:

```
if (Uslov)
{
    // blok naredbi koji se izvršava kada je uslov tačan
}
Else
{
    // blok naredbi koji se izvršava kada je uslov nije
    tačan
}
```

Na primer:

```
If (a == 2)
{
    neki kod...
}
```

Deo else je opcioni i ne mora se navoditi.

Uslov može biti kompozitni, složen od dva ili više pojedinačnih uslova. U tom slučaju je neophodno ove pojedinačne uslove spojiti odgovarajućim logičkim operatorima.

Operatori AND (i) i OR (ili) se sintaksno označavaju sa && i || respektivno.

Na primer:

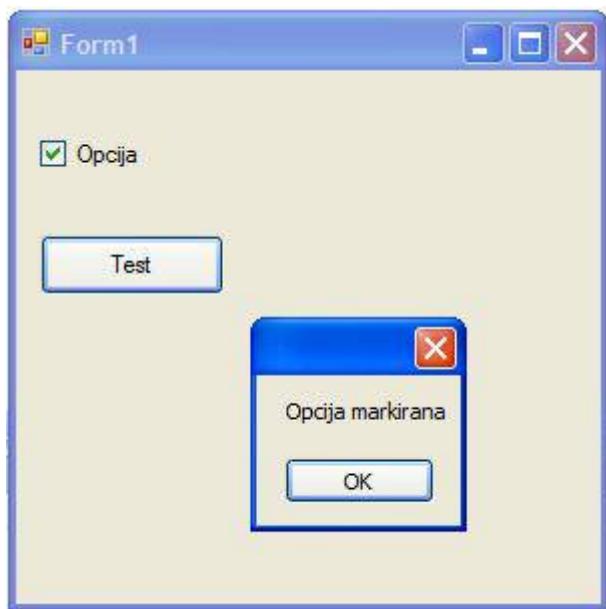
```
If (a==5 || b ==7)
```

uslov je tačan ako je tačno a=5 ili b=7 ili su oba uslova tačna.

Sada duplim klikom na dugme otvorite prozor za pisanje koda. Pošto nismo menjali inicijalno postavljena imena kontrola, CheckBox kontrola se zove checkBox1. Upišite sledeći segment koda u button1_Click proceduri:

```
if (checkBox1.Checked == true)
{
    MessageBox.Show("Opcija markirana");
}
else {
    MessageBox.Show("Opcija nije markirana");
}
```

Pokrenite aplikaciju (F5 funkcijiški taster) i isprobajte je. Zavisno od stanja CheckBox kontrole treba da dobijete odgovarajuću poruku:



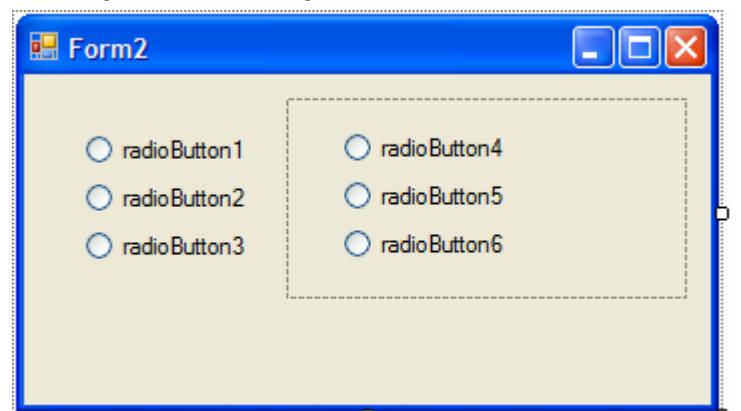
Radio Button kontrola

Ova kontrola je slična CheckBox kontroli, ali u ovom slučaju možemo markirati samo jednu kontrolu u okviru iste grupe. Otud i naziv "radio dugme", setite se starih radio aparata gde se preko mehaničkih dugmadi birao frekventni opseg. Pritiskom na bilo koje dugme, predhodno uključeno se isključuje – samo jedno može biti upaljeno.

Pod grupom radio dugmića se smatraju svi koji su postavljeni na isti kontejner kontrola. Očigledni kontejner kontrola je formular, ali ih ima još nekoliko na raspolaganju.

Na formu postavite tri Radio Button kontrole. One su na istom kontejneru – formular, te su povezane i međusobno se isključuju.

Zatim na formu postavite kontrolu Panel. Ona predstavlja drugi kontejner na koji ćemo staviti još tri radio dugmeta. Konačni izgled formulara:



Sada pokrenite aplikaciju i testirajte je. Primetićete da možete markirati samo jedno radio dugme u svakoj grupi.

U praksi se Panel kontrola najčešće koristi za grupisanje radio dugmadi.

Bitna svojstva su identična kao kod CheckBox kontrole: Checked i Text sa istim vrednostima i funkcijom.

Korišćenje je takođe identično kao kod CheckBox kontrole. U kodu se proveravaju statusi radio kontrola i zavisno od toga se vrši grananje u aplikaciji.

List Box kontrola

Ovo je složenija kontrola, koja se sastoji od više stavki. Na primer, može predstavljati listu predmeta u školi. Korisnik može izabrati predmet i na primer upisati ocene i slično.

Liste se mogu "puniti" za vreme dizajna, ali je ipak najčešći slučaj punjenje u vreme izvršavanja i to iz baze podataka. Za sada ćemo je napuniti pomoću jednostavne petlje iz koda.

Ako koristimo objektno orijentisaniu terminologiju, možemo reći da je lista kolekcija stavki tipa Object.

Kolekcija označava skup objekata istog tipa. Svaki član kolekcije ima svoj jedinstveni indeks koji počinje od broja 0 do ukupnog broja stavki u kolekciji umanjenog za jedan.

Bitno je razumeti da kolekcija ima svoja specifična svojstva i metode, ali takođe i svaki objekat u kolekciji.

Na primer svaka kolekcija ima svojstvo Count koje je samo za čitanje i daje broj objekata u kolekciji. Kolekcije imaju metode kojima se može dodavati nov objekat u kolekciju (Add), obrisati pojedinačni objekat (RemoveAt) ili obrisati sve članove kolekcije (Clear). Svojstva i metode pojedinačnih objekata kolekcije zavise od tipa objekata.

Ako postoji kolekcija pod nazivom "MojaKolekcija", metodima i svojstvima te kolekcije pristupamo na sledeći način:

MojaKolekcija.Add (...)

MojeKolekcija.Clear ()

i slično

Pojedinačnim objektima kolekcije se pristupa preko indeksa objekta:

MojaKolekcija[5].Text

Indeksi objekata – članova kolekcije idu od nula do MojaKolekcija.Count – 1

U List Box kontroli ova kolekcija se zove Items i sastoji se od članova pod nazivom Item.

Sve kolekcije .NET Framework-a poštuju istu nomenklaturu: kolekcija je imenica u množini a svaki član kolekcije je ista imenica u jednini. Na primer kolekcija Tables se sastoji od objekata tipa Table.

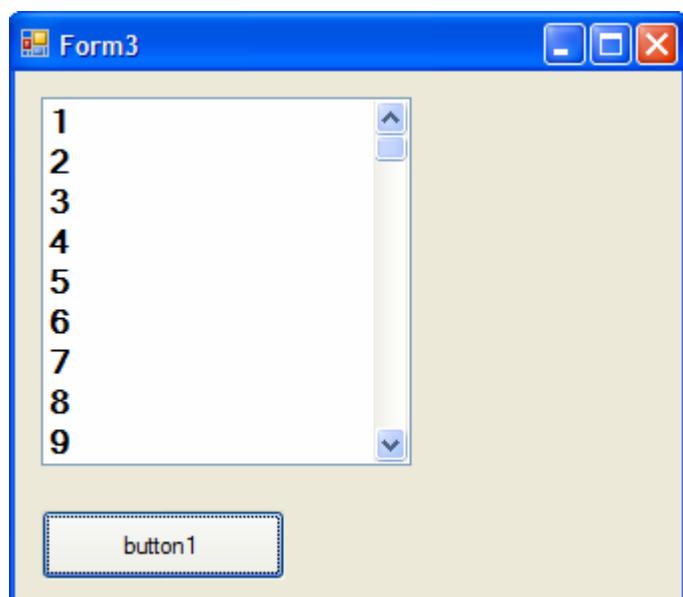
Na formu postavite List Box i jednu Button kontrolu. Ostavićemo inicijalne nazive ovih kontrola listBox1 i button1.

Želimo da klikom na dugme napunimo listu brojevima od 1 do 100. Koristimo metodu Add kolekcije Items gde je ulazni parametar generički tip Object što znači da možemo "podmetnuti" varijable numeričkog ili string tipa – svejedno, sve su interno nasleđene od istog Object tipa. Duplim klikom na dugme otvorite prozor za pisanje koda:

```
int i;
for (i = 1; i <= 100; i++)
{
    listBox1.Items.Add(i);
}
```

Pokrenite primer i testirajte ga. Primetićete da svaki put kada pritisnete dugme dodajete još 100 stavki u listu.

Ako želimo da pre dodavanja obrišemo sve stavke (ako ih ima), dodaćemo još jednu liniju koda pre petlje koja briše sadržaj kolekcije Items:



```
int i;
listBox1.Items.Clear();
for (i = 1; i <= 100; i++)
{
    listBox1.Items.Add(i);
}
```

Sada, pre punjenja liste uvek brišemo sve stavke.

Ako želimo da pročitamo vrednost izabrane stavke iz liste, postoje dva načina:

- jednostavno pročitamo svojstvo Text: `listBox1.Text`
- pomoću svojstva SelectedItem koja vraća Object prezentaciju izabrane stavke: `listBox1.SelectedItem`

Brisanje pojedinačne stavke iz liste se vrši metodom RemoveAt kolekcije Items. Na primer ako želimo da obrišemo treću stavku u listi:

```
listBox1.Items.RemoveAt(2);
```

Obratite pažnju da je indeks treće stavke = 2 jer se indeksi broje od nule.

Svojstvo SelectedIndex pokazuje indeks izabrane stavke iz liste. U pitanju je tip int, a ako ni jedna stavka nije izabrana vrednost SelectedIndex svojstva je jednaka -1.

Ovo se koristi kao indikacija da korisnik nije izabrao ni jednu stavku iz liste.

SelectedIndex svojstvo je predviđeno i za čitanje i za pisanje, tako da ako iz koda želimo da izaberemo na primer petu stavku u listi:

```
listBox1.SelectedIndex = 4;
```

U narednim nastavcima ovog serijala ćemo se upoznati sa još nekoliko kontrola i polako preći na korišćenje baze podataka u C# aplikacijama.

Mala škola programiranja C# (9)

Kreiranje menija u aplikaciji

Posle letnjih odmora nastavljamo sa malom školom programiranja u C# programskom jeziku. Nadam se da smo se svi odmorili i da smo spremni za učenje.

Rad sa menijem

Gotovo svaka Windows aplikacija ima meni kao davno ustanovljeni vid interakcije sa korisnikom. Prednosti menija su mogućnost hijerarhijske organizacije koje odgovaraju vašoj aplikaciji uz minimalno zauzeće prostora.

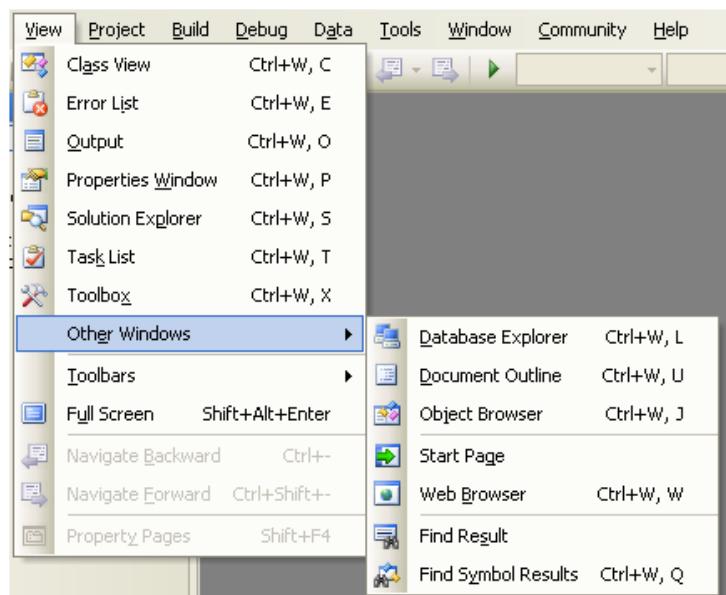
Meni se obično nalazi pri vrhu forme, ali to ne mora biti slučaj. Možete ga postaviti na dno ekrana, pa čak i po vertikali ako vam to odgovara.

Dodatnu snagu menija predstavlja mogućnost dinamičkog dodavanja ili sakrivanja stavki čime meni prati korisnika zavisno od akcija koje se događaju u aplikaciji.

U Visual Studiju kreiranje menija je lako, može se reći zabavno i značajno je unapređeno u odnosu na sada već prastari ali svojevremeno veoma popularni Microsoft Visual Basic 6.

Prilikom kreiranja menija razlikujemo horizontalnu liniju i vertikalne delove menija koji se prikazuju kada korisnik klikne na stavku u horizontalnoj liniji. Vertikalna linija može imati stavki na više nivoa hijerarhije.

Na sledećoj slici vidite View meni iz radnog okruženja Visual C# Express edition.



Osim teksta stavka menija može imati sličicu (eng. Icon) i definisanu skraćenicu sa tastature (eng. Shortcut).

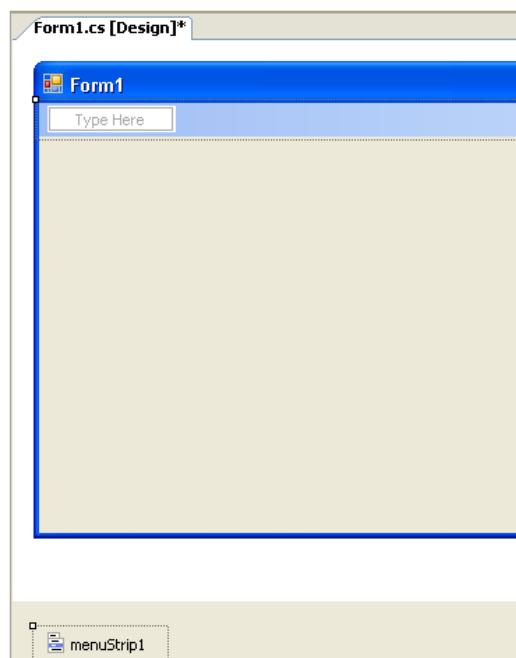
Kreiranje menija

Pokrenite Visual C# Express razvojno okruženje i kreirajte novu Windows aplikaciju. Meni ćemo kreirati na inicijalnoj formi (Form1).

U paleti sa alatkama (eng. Toolbox) pronađite sekciju „Menus & Toolbars“. U okviru nje pronađite kontrolu „MenuStrip“ koja daje potrebne funkcionalnosti menija na formi.



Prevucite je i pustite na formu - nije važno gde, jer se meni automatski pozicionira na vrh forme, a sama kontrola u specijalnom prozoru ispod forme kao što je prikazano na sledećoj slici.

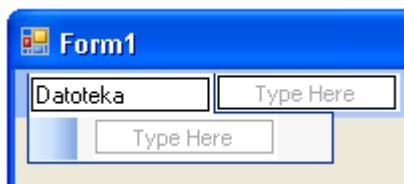


Ako želite da kreirate meni, prvo morate da kliknete na kontrolu na dnu forme koja je dobila inicijalno ime „menuStrip1”. Tada se u dizajneru prikazuje meni na vrhu forme.

Zapazite tekst „Type Here” koji dovoljno govori sam za sebe. Napravićemo deo menija koji je manje-više standardan za većinu aplikacija.

Tamo gde piše „Type Here” ukucajte „Datoteka” (bez znakova navoda) ili „File” ako želite meni na engleskom jeziku.

Čim počnete da kucate odmah će se ispod i desno od ove stavke otvoriti mogućnost za dalji unos stavki menija, horizontalno i vertikalno:



Verovatno ste primetili da svaki meni ima pridružen takozvani „hot key”. Slovo ili broj koji uz pritisnut „Alt” specijalni taster predstavlja skraćenicu za izbor stavke u horizontalnom delu menija.

Ova funkcionalnost se jednostavno definiše – potrebno je ispred karaktera koji treba da bude „hot key” ukucati karakter &. Ako želimo da „hot key” za meni „Datoteka” bude kombinacija Alt+D jednostavno ukucajte „&Datoteka”.

Karakter & se ne prikazuje, već je ispisano „Datoteka”. Podvučeno slovo označava skraćenicu sa tastature. Ovo se i vidi čim pritisnute Enter taster čime završavate unos teksta u stavci menija.

Nastavljamo sa menjem „Datoteka” tako što ćemo ispod njega napraviti nekoliko stavki. Ukucajte jedno ispod drugog, redom:

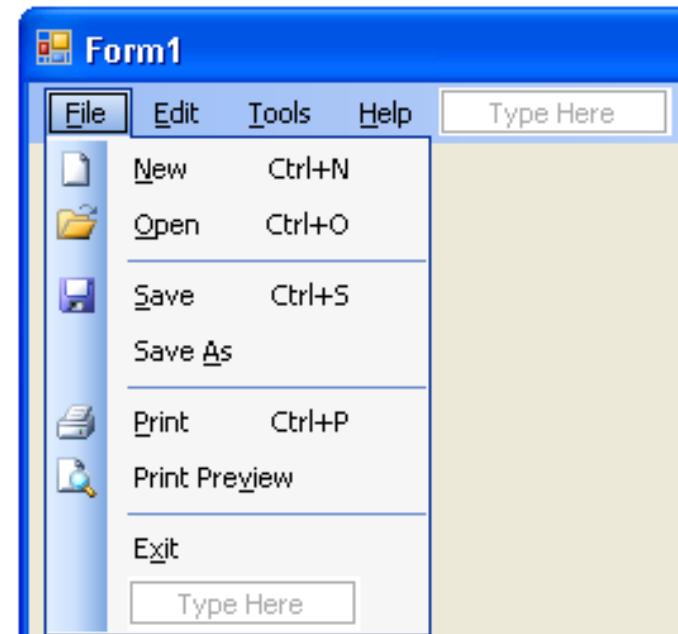
- &Otvori
- &Snimi
- Snimi k&ao
- -
- &Kraj rada

Ako za tekst stavke unesete samo znak – (minus), on ima specijalno značenje: u meniju se prikazuje horizontalni separator. Korisno kada treba da vizuelno razdvojite različite funkcionalne grupe u okviru vertikalnog menija. Na kraju naš meni treba da izgleda kao na sledećoj slici:



Redom bi dalje popunjavali ostatak menija, zavisno od funkcionalnosti aplikacije.

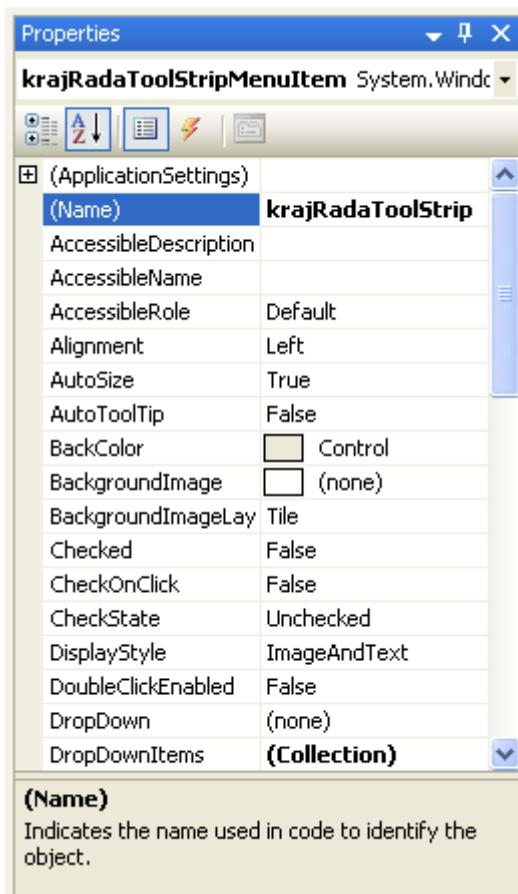
Primer menija na formi koji je popunjeno standarnim stavkama (na engleskom jeziku) možete videti na sledećoj slici:



Programiranje menija:

Svaku stavku menija je najbolje tretirati kao komadno dugme (eng. Button). Programiranje se vrši u Click događaju stavke menija.

U primeru koji smo započeli kliknite samo jednom na stavku menija „Kraj rada“. Svaka stavka menija ima svoj set svojstava koja su vidljiva u standardnom prozoru za prikaz svojstava objekta - „Properties“ prozoru:



Kao i kod svake druge kontrole ključno svojstvo je „Name“ koje je automatski generisano – u našem slučaju „krajRadaToolStripMenultem“.

Svojstvo „Text“ definiše tekst stavke menija i njega smo podešavali iz dizajnera.

Sada uradite dupli klik na stavku menija „Kraj rada“, čime je otvoren Click događaj u prozoru za pisanje koda. Kada korisnik klikne na „Kraj rada“ želimo da zauzavimo izvršavanje programa. Upišite sledeću liniju koda:

```
Application.Exit();
```

Application je referenca na aplikaciju koju kreiramo. Poseduje metod Exit koji izvršava bezuslovni prekid rada aplikacije i oslobađa sve resurse koje je aplikacija zauzela (memoriju, datoteke, baze podataka i tako dalje...)

Pokrenite aplikaciju i mišem kliknite na „Kraj rada“.

Pokušajte takođe i pomoću skraćenica sa tastature: prvo Alt+D kako bi otvorili meni „Datoteka“, a potom samo taster K kako bi aktivirali stavku „Kraj rada“.

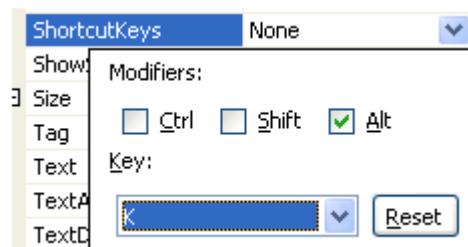
Skraćenice

Da li je moguće definisati skraćenicu sa tastature koja će, na primer, direktno aktivirati stavku „Kraj rada“?

Naravno da je moguće, čak se može definisati kombinacija bilo kog specijalnog tastera (Control, Alt ili Shift) i običnog karaktera. Takođe možete iskoristiti funkcione i druge specijalne karaktere sa tastature.

Zaustavite program ako je pokrenut i ponovo jednim klikom miša izaberite stavku „Kraj rada“.

Potom u prozoru svojstava (eng. Properties) pronađite svojstvo „ShortcutKeys“. Otvorite padajuću listu pored ovog svojstva i trebalo bi odmah sve da bude jasno:



Markirajte Alt i iz padajuće liste izaberite slovo K kao što je prikazano na gornjoj slici. Sada kombinacija Alt+K završava našu aplikaciju. Pokrenite aplikaciju i probajte.

Primite da se uz stavku menija prikazuje i njena skraćenica. Ovo se podešava svojstvom „ShowShortcutKeys“ koja može imati vrednosti True (da) ili False (ne).



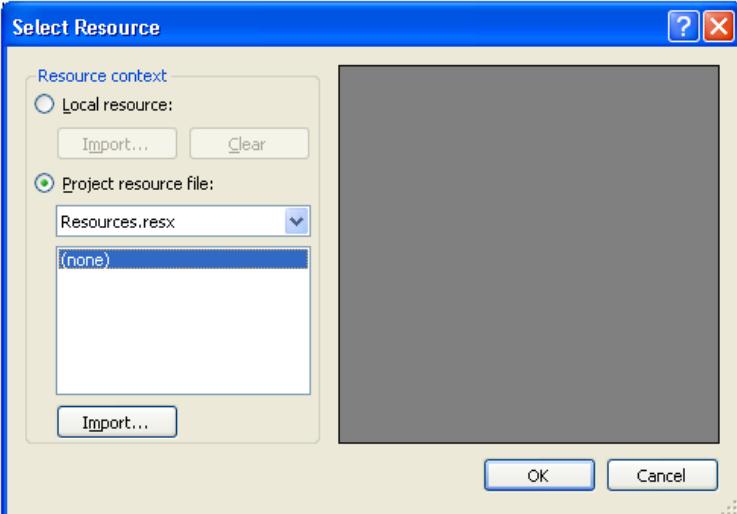
Slike

Svaka stavka menija (uključujući i stavke u horizontalnoj liniji) može imati i sličicu (eng. Icon) koja doprinosi lepšem i razumljivijem korisničkom interfejsu.

Sličice koje se koriste za ovu namenu su najčešće bmp ili jpeg formata, dimenzija 16x16 piksela. Pogledajte na Internetu i načiđete veliki broj kolekcija sličica za besplatno preuzimanje. Ako ste vešti možete ih napraviti i sami.

Pomoću svojstva „Image“ možete da izaberete sliku sa vašeg računara i dodelite je stavci vašeg menija.

Kliknite na malo dugme pored ovog svojstva i dobi-



ćete sledeći dijalog:

U ovom dijalogu se dodaju resursi aplikaciji. Oni osim slika mogu biti i **zvučne datoteke**, animacije kao i druge binarne datoteke koje koristite u aplikaciji.

Kliknite na dugme „Import“ i sa diska vašeg računara izaberite sliku koju ste pripremili. Na kraju kliknite na dugme „OK“. Ako je slika veća od 16x16 piksela, biće automatski skalirana na tu veličinu.

Korisna je i mogućnost da pored svake stavke menija imate znak za overu (eng. Check).

Svojstvo Checked koje može imati vrednosti True ili False obezbeđuje upravo ovo.

Izaberite stavku „Snimi“ u svojstvima pronađite Checked i postavite ga na True. Sa leve strane ove stavke se pojavljuje znak za overu.

U praksi se znak za overu prikazuje ili ne zavisno od logike programskog toka aplikacije i akcija korisnika. Na primer, linija koda:

```
snimiToolStripMenuItem.Checked = true;
```

će za vreme izvršavanja programa postaviti znak za overu pored stavke menija.

Takođe, u praksi se često koristi dinamičko prikazivanje ili skrivanje stavke menija ili cele grupe, opet zavisno od logike aplikacije. Za ovo imamo dve mogućnosti, odnosno svojstva:

1. Svojstvo Enabled koje kada je postavljeno na False, prikazuje stavku menija u sivoj boji i korisnik ne može da klikne na nju. Korisno ako želite da korisnik zna da stavka menija postoji, ali trenutno nije dostupna.
2. Restriktivnije svojstvo Visible (koje imaju sve kontrole) postavljeno na False čini da se stavka menija uopšte ne vidi.

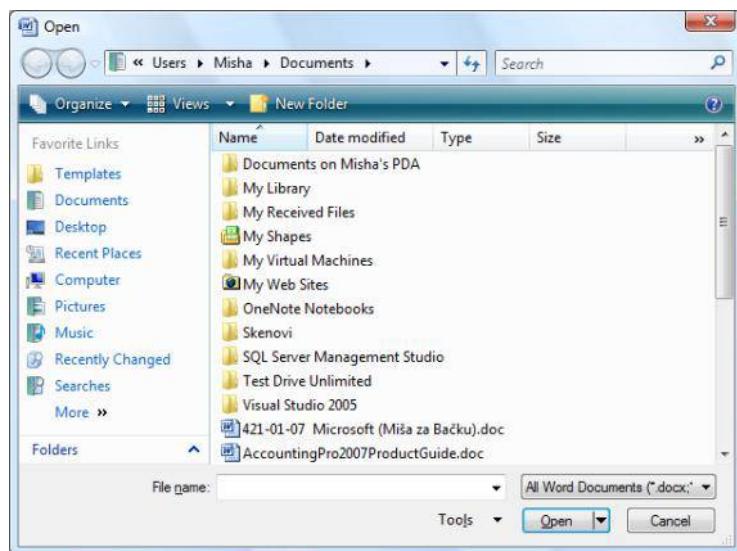
Mala škola programiranja C# (10)

Rad sa standardnim dijalozima i datotekama

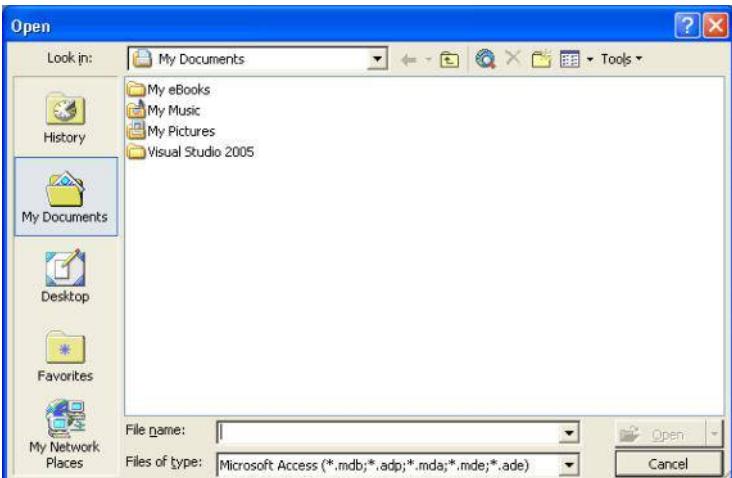
Većina Windows aplikacija ima potrebe da manipuliše različitim datotekama; tekstualne datoteke, slike, itd. u smislu čitanja i pisanja sadržaja.

U ovom nastavku male škole programiranja za C# ćemo se posvetiti upravo ovome. Manipulacija datotekama obavezno uključuje i standardne dijaloge za izbor i čuvanje sadržaja. Oni se otvaraju izborom opcija Snimi (eng. Save) i Otvari (eng. Open).

Na primer, u Microsoft Wordu dijalog za otvaranje dokumenta je prikazan na sledećoj slici:



Zavisno od verzije operativnog sistema, zavisi izgled i funkcionalnost ovih dijaloga. Na gornjoj slici je prikazan Open dijalog u operativnom sistemu Windows Vista. U Windows XP operativnom sistemu on izgleda malo drugačije:



U svakom slučaju osnovne funkcionalnosti su iste:

Moguće je izabrati datoteku, prikazane su samo datoteke koje zadovoljavaju određeni kriterijum (samo doc ekstenzije na prvoj, odnosno mdb na drugoj slici) uz mogućnost da se ručno unese ime datoteke - File name polje.

Dobra vest je što nema potrebe ručno programirati ove dijaloge. Oni su sistemski i postoje u svakoj verziji Windowsa od kada on postoji. Čak, ako na primer kreirate aplikaciju u Windowsu XP sa ovim dijalozima i kopirate izvršnu datoteku na Windows Vistu videćete Vistine dijaloge bez ikakve izmene aplikacije.

Korišćenje ovih dijaloga je veoma jednostavno, direktno i potpuno podržano u Visual C# Express Edition.

Korak-po-korak sistem za korišćenje ovih dijaloga:

1. postaviti odgovarajuću kontrolu na formular
2. postaviti potrebna svojstva kontrole (naslov, početni direktorijum, filter za ekstenzije datoteka...)
3. otvaranje dijaloga
4. po zatvaranju dijaloga proveriti da li je korisnik izabrao datoteku ili kliknuo na dugme Odustani (eng. Cancel)
5. ako je izabrana datoteka, iz svojstva FileName čitamo putanju i ime datoteke i dalje radimo sa njom

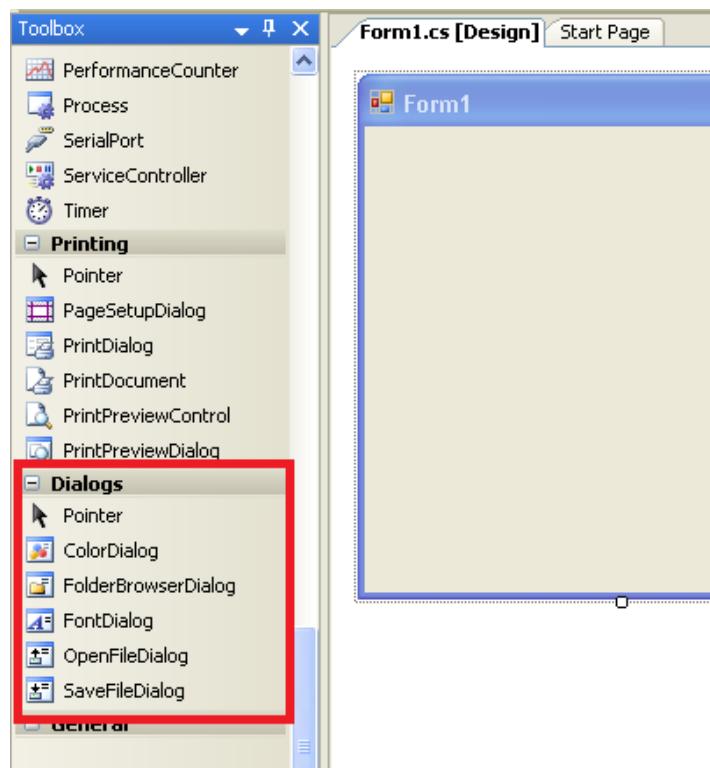
Napravićemo jednostavan primer koji pokreće Open dijalog i ime izabrane datoteke ispisuje u naslovu formulara.

Pokrenite Visual C# Express Edition i u početnom ekranu kliknite na link Create Project. U dijalogu "New Project" izaberite "Windows Application" i kliknite na dugme "OK"

Posle nekoliko sekundi projekt je kreiran i prikazan je početni formular. Na dalje radimo po gore prikazanim koracima.

1. postaviti odgovarajuću kontrolu na formular

U paleti sa alatkama (sa leve strane razvojnog okruženja), pronađite sekciju "Dialogs" kao što je prikazano na sledećoj slici:



U ovoj sekciji se nalaze svi standardni Windows dijalozi. Osim pomenutih, tu su dijalozi za izbor boje, pregled direktorijuma i izbor fonta.

Na formular postavite (prevuci i pusti postupak) kontrolu OpenFileDialog. Kontrola se prikazuje na traci ispod formulara i inicijalno je dobila naziv (Name svojstvo) openFileDialog1.

2. Postaviti potrebna svojstva kontrole (naslov, početni direktorijum, filter za ekstenzije datoteka)

Kada pokrenemo dijalog za otvaranje datoteke, uobičajeno je da u njemu vidimo samo datoteke sa određenom ekstenzijom koju naša aplikacija prepoznaje. Obično se za svaki slučaj dodaje i opcija za prikaz svih datoteka - filter *.* sigurno ste do sada u ovakvim dijalozima videli opciju "All files" (sve datoteke).

Filter Open dijaloga se postavlja pomoću istoimenog svojstva Filter. Ovo svojstvo je tipa string i zahteva specifično formatiranje. Na primer, ako ovo svojstvo postavimo na:

Tekst datoteke|*.txt|Sve datoteke|*.*

napravili smo dva filtera. Prvi sa nazivom "Tekst datoteke" koji prikazuje sve datoteke sa ekstenzijom txt, i drugi sa nazivom "Sve datoteke" koji prikazuje upravo to: sve datoteke sa bilo kojom ekstenzijom.

Iz ovoga se vidi da se svaki filter sastoji od dva dela. Naziv filtera i kriterijum za prikaz datoteka. Separator je vertikalna crta. **Obratite pažnju da na kraju ne treba staviti vertikalnu crtu.**

Dozvoljene su i sledeće kombinacije:

Slike|*.jpg;*.gif;*.bmp|Video|*.mpg;*.avi;*.wmv

U ovom slučaju takođe imamo dva filtera, Slike i Video ali svaki od njih će prikazati datoteke sa više ekstenzija. Separator između ekstenzija istog filtera je karakter **tačka**-zarez.

U našem primeru, svojstvo Filter openFileDialog1 kontrole postavite na:

Tekst datoteke|*.txt|Sve datoteke|*.*

Obično se postavlja i naslov dijaloga koji se definiše pomoću svojstva Title. Pošto u našem primeru treba otvoriti tekstualne datoteke postavite ovo svojstvo na Otvori tekstualnu datoteku.

Opciono možemo postaviti naziv direktorijuma čiji će sadržaj biti prikazan odmah po otvaranju dijaloga. Ako želimo da se odmah prikaže koren diska C: u svojstvo InitialDirectory upišite C:\.

Sada smo pripremili dijalog za otvaranje.

NAPOMENA:

Sva pomenuta svojstva se mogu postaviti kako u vreme dizajniranja (kao što smo mi uradili), tako i u vreme izvršavanja programa.

3. Otvaranje dijaloga

Dijalog poseduje metodu ShowDialog koja kao rezultat izvršavanja vraća akciju korisnika u dijalogu. Ova akcija predstavlja enumeraciju pod nazivom DialogResult. Nama su od interesa samo dve opcije:

DialogResult.OK - korisnik je izabrao datoteku

DialogResult.Cancel - korisnik je zatvorio dialog (kliknuo na Cancel dugme)

Proverom ove povratne vrednosti znamo da li je datoteka izabrana i dalje nastavljamo rad sa njom.

Na formular postavite jedno dugme (Button kontrola) i uradite dupli klik mišem na njega kako bi napisali par linija kôda.

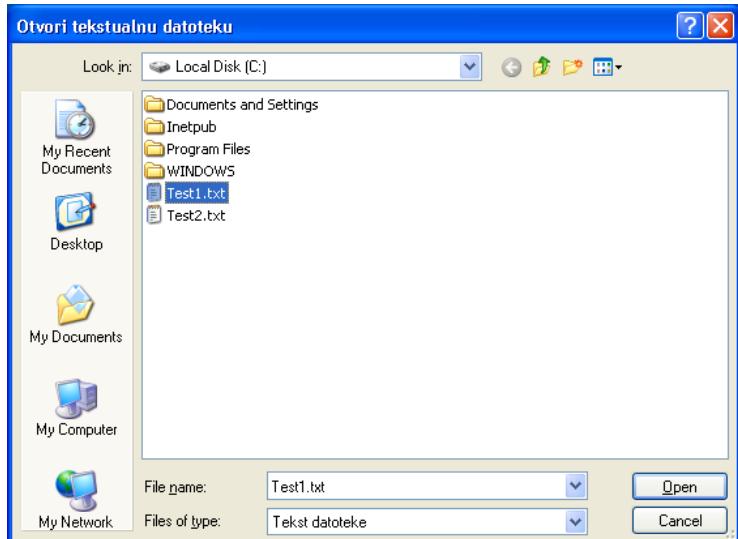
```
private void button1_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        this.Text = openFileDialog1.FileName;
    }
}
```

Pomoću openFileDialog1.ShowDialog() otvaramo dijalog i odmah proveravamo njegovu povratnu vrednost. Ako je ona jednaka DialogResult.OK, znači da je korisnik izabrao datoteku i kliknuo na dugme “**Open**” u dijalušu.

U tom slučaju u svojstvu FileName se nalazi ime izabrane datoteke sa kompletom putanjom. U naslov formulara (this.Text) upisujemo putanju i ime izabrane datoteke.

Dakle svojstvo FileName je ključno, jer definiše lokaciju izabrane datoteke.

Pokrenite program (F5 funkcijski taster) i kliknite na dugme. Obratite pažnju na dva filtera koja smo kreirali i proverite funkcionisanje oba. Na svom računaru pronađite odgovarajuću datoteku i kliknite na dugme “Open” na dijalušu. Nakon ovoga u naslovu forme će biti ispisana



Prikaz sadržaja izabrane datoteke

Otvaranje dijaloga i izbor datoteke, naravno ne otvara niti prikazuje njen sadržaj. Sledeći korak je da zaista prikažemo sadržaj tekstualne datoteke.

Za otvaranje, čitanje i pisanje po datotekama je zadužen set klasa u imenovanom prostoru (eng. NameSpace) System.IO

Za čitanje datoteka koristi se klasa StreamReader kojoj se u konstruktoru zadaje putanja do datoteke. Tačno ono što već imamo u FileName svojstvu dijaloga.

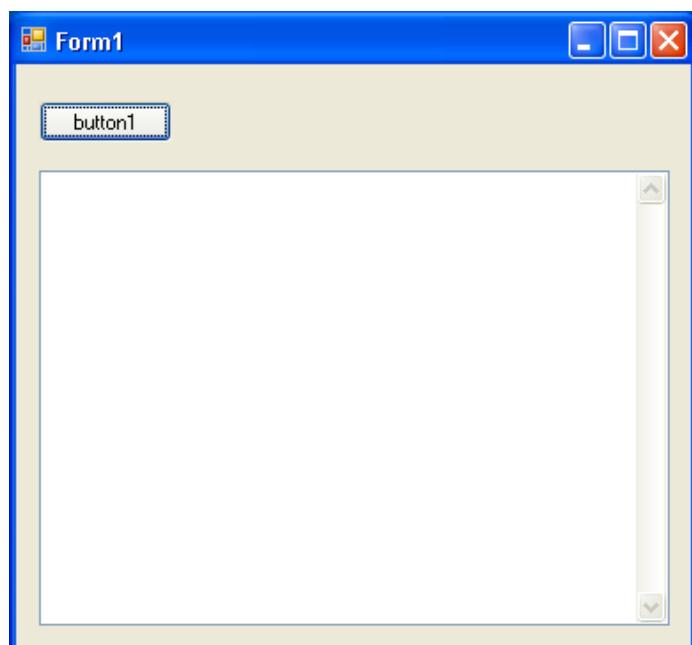
Metodom ReadToEnd StreamReader objekta čita se kompletan sadržaj zadate datoteke, i ova metoda vraća tip string koji ćemo prikazati u kontroli TextBox.

Zaustavite aplikaciju i na formular postavite TextBox kontrolu u kojoj ćemo ispisati sadržaj tekstualne datoteke. Kontrola je inicijalno dobila ime (Name svojstvo) textBox1. Postavite joj sledeća svojstva na prikazane vrednosti:

Multiline: True

ScrollBars: Vertical

Formular sada treba da izgleda kao na sledećoj slici:



Uradite dupli klik na dugme, kako bi modifikovali kôd kao što je prikazano na sledećoj strani:

```

private void button1_Click(object sender, EventArgs e)
{
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    this.Text = openFileDialog1.FileName;
    System.IO.StreamReader sr =
    new StreamReader
        (openFileDialog1.FileName);
    textBox1.Text = sr.ReadToEnd();
    sr.Close();
}
}

```

Inicijalizujemo objekat sr tipa System.IO.StreamReader i u konstruktoru postavljamo ime datoteke koju je korisnik izabrao u dijalogu.

Metodom ReadToEnd() čitamo kompletan sadržaj datoteke i postavljamo ga u Text svojstvo kontrole textBox1.

Na kraju treba obavezno zatvoriti StreamReader objekat, što se radi metodom Close().

UPOZORENJE:

Ako StreamReader ne zatvorite metodom Close(), datoteka koja se čita ostaje **zaključana** (eng. Locked) tako da ni jedna druga aplikacija ne može da je otvari. Kaže se da StreamReader datoteku otvara u ekskluzivnom režimu.

Međutim, ako regularno zatvorite svoju aplikaciju, automatski se "otključavaju" sve otvorene datoteke.



Provjerajte sami!

Pokušajte da proširite primer tako što ćete omogućiti korisniku da sadržaj textBox1 kontrole snimi kao novu datoteku.

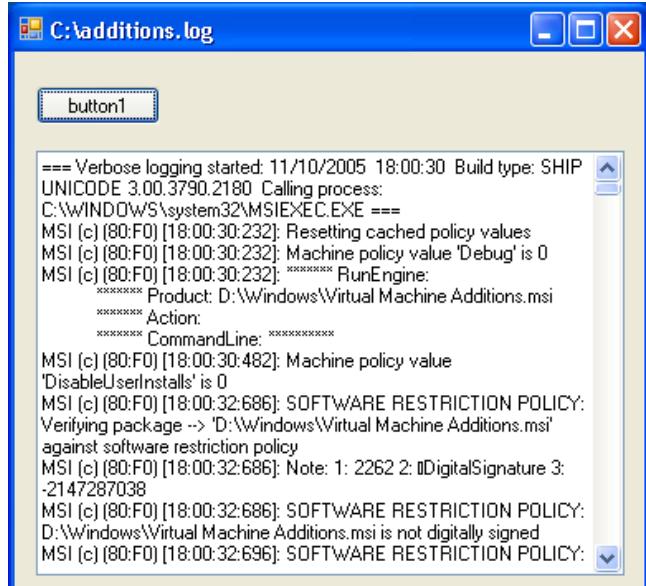
Treba dodati još jedno dugme (Button) na formular u kome ćete za Click događaj napisati sav potrebnii kôd.

Ovog puta koristimo SaveFileDialog kontrolu. Svojs-tva i upotreba je identična kao u OpenFileDialog kontroli koju smo koristili u primeru.

Klasa StreamWriter nam daje sve potrebne funkcionalnosti. U konstruktoru se takođe navodi puna putanja i ime datoteke, ali sada je to datoteka koja ne postoji, već će je kreirati StreamWriter objekat.

Upis sadržaja u datoteku se vrši metodom Write koja kao ulazni argument ima tekst koji treba da upiše u datoteku.

Takođe je potrebno na kraju zatvoriti StreamWriter objekat pomoću metode Close().



Mala škola programiranja C# (11)

Greške u programiranju i obrada grešaka

U februarskom broju časopisa i četvrtom nastavku škole programiranja smo prvi put pomenuli i ukratko objasnili greške u C# programiranju. Sada je vreme da detaljnije obradimo ovu problematiku.

Greške u programiranju se događaju. Nije pitanje da li će se u vašoj aplikaciji dogoditi greška, već kada će se ona dogoditi. Greške u radu programa su činjenice koje treba prihvati i na odgovarajući način reagovati.

Teoretski govoreći, postoje tri grupe grešaka na koje se nailazi u svakom programiranju:

1. Sintaksne greške

Nastaju kada pogrešite u sintaksi programskog jezika, korišćenju neke klase i slično. Ove greške nisu "opasne" jer aplikacija koja ih sadrži jednostavno ne može da se prevede (kompajlira), pa samim tim ni pokrene.

2. Greške u vreme izvršavanja (run-time errors)

Aplikacija nema sintaksnih grešaka, uspešno je kompajlirana i pokrenuta. Međutim, u toku rada aplikacije dogodila se greška (na primer deljenje sa nulom) i ako ništa nismo preduzeli, aplikacija će prestati sa radom. Ovo su najčešće greške i predstavljaju fokus ovog članka.

3. Logičke greške

Aplikacija je uspešno kompajlirana, prilikom rada ne prijavljuje nikakve greške niti prestaje sa radom. Sve je u redu, ali vaš program "samo" ne radi ono što treba da radi. Vraća pogrešne rezultate, ne upisuje podatke i tome slično. Ovo su verovatno greške koje je najteže otkriti jer su posledice lošeg dizajna aplikacije.

1. Sintaksne greške

Radno okruženje Visual Studio C# Express pruža puno pomoći za otkrivanje i uklanjanje sintaksnih grešaka. Već prilikom pisanja koda, sve problematične konstrukcije će biti jasno označene, sa ponudom za pomoći ili automatsko ispravljanje gde god je to moguće.

Pokrenite Visual C# Express i kreirajte novu Windows aplikaciju. Na inicijalnu formu postavite jedno komandno dugme u uradite dupli klik. Kao što smo videli ranije, automatski se otvara prozor za pisanje koda koji obrađuje Click događaj komandnog dugmeta.

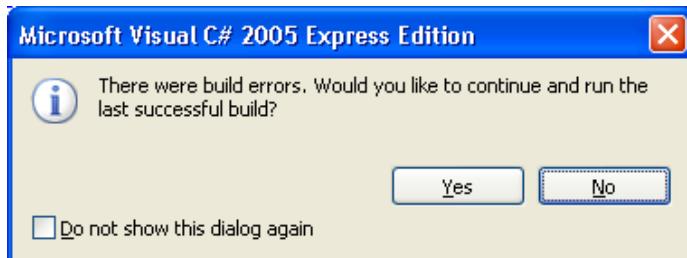
Ovde ćemo namerno napraviti sintaksnu grešku i videti kako je ona prikazana u razvojnom okruženju. Otkucajte sledeću liniju koda:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox("TEST");
}
```

Simulirali smo grešku tako što smo izostavili metodu Show, tako da bi ispravna linija trebala da glasi:

```
MessageBox.Show("TEST");
```

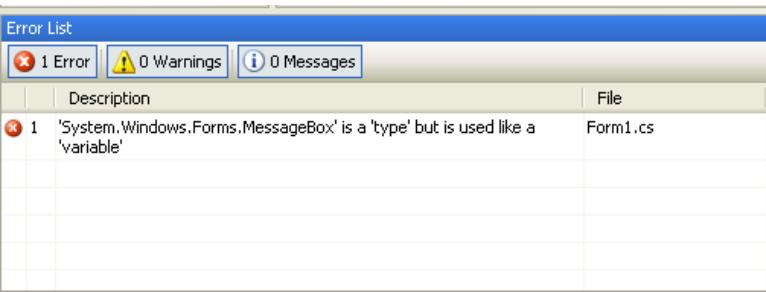
Sada pokrenite aplikaciju (F5 funkcijski taster). Obratite pažnju na pitanje koju dobijate:



Ako odgovorite potvrđno na ovo pitanje, biće pokrenuta predhodno uspešno kompajlirana verzija aplikacije (ako postoji). Međutim ovo obično nema smisla, tako da treba kliknuti na "No", čime je obustavljeno pokretanje aplikacije. Ako sada pogledate kod, primetićete da je problematična linija podvučena talasastom plavom linijom, kao što je prikazano na sledećoj slici:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox("TEST");
}
```

Takođe pogledajte prozor "Error list" (lista grešaka) koji se inicijalno nalazi na dnu razvojnog okruženja.



U ovom prozoru se nalazi lista svih sintaksnih grešaka kao i lista grešaka koje se dogode u vreme izvršavanja. Za svaku grešku imamo opis, u kojoj datoteci se nalazi, broj linije i kolone, kao i naziv projekta.

Ako uradite dupli klik na grešku u ovom prozoru, automatski se otvara datoteka gde se ona nalazi, a kurzor se pozicionira na problematičnu liniju. Ostaje da ispravite grešku i ponovo pokrenete program.

2. Greške u vreme izvršavanja (run-time errors)

Greške koje ne možemo izbeći i koje su rezultat unosa korisnika, infrastrukture, mreže i drugih računarskih resursa. Početnici u programiranju obično ne shvataju šta sve može krenuti naopako, iako je program besprekorno napisan.

Na primer, napisan je program koji čita podatke iz baze podataka i prikazuje ih na formularu. Baza podataka je na drugom računaru u mreži. Ovde u praksi može nastati mnogo problema gde direktni razlog nije sama aplikacija. Ako iz nekog razloga mreža nije dostupna zbog hardverskih problema (a njih zaista može biti puno), aplikacije će krahirati. Možda je mreža u redu, ali database server nije pokrenut. Možda je u redu i mreža i database server, ali je baza podataka oštećena, ili ima previše korisnika, ili je baš u tom momentu baza off-line jer administrator radi svoj posao...

Dobili ste sliku.

Strogo govoreći, svaka linija koda u programu može da generiše grešku. U praksi, nećemo za svaku liniju postavljati obradu grešaka, već samo tamo gde postoji realna verovatnoća da greška nastane.

U fazi testiranja aplikacije se upravo i otkrivaju takva mesta koja možda u početku nisu očigledna.

.NET Framework ima snažnu podršku i mogućnosti za upravljanje greškama u vreme izvršavanja programa. U C# programskom jeziku je usvojena klasična struktura za obradu grešaka koja postoji u C / C++ programskim jezicima. Sintaksa je sledeća:

```
try
{
    // linije koda gde može da nastane greška
}
catch (Exception e1)
{
    // obrada greške
}
finally
{
    // deo koji se uvek izvršava
}
```

Postoje tri bloka od kojih je poslednji (`finally`) opcional.

U bloku `try` treba da se nalazi jedna ili više linija koda gde se mogu pojaviti greške u vreme izvršavanja aplikacije.

U bloku `catch` se nalazi kod koji obrađuje greške (eng. error handler). Ako se na bilo kojoj liniji koda u `try` bloku dogodi greška, tok programa se automatski preusmerava na prvu naredbu u `catch` bloku.

Ovde možete probati da ispravite grešku, ili samo informišete korisnika i prikažete mu poruku greške.

Može postojati više `catch` blokova koji "hvataju" različite grupe grešaka. Na primer, ako otvarate konekciju ka bazi podataka, prvo bi mogli da uhvatimo probleme u mreži, zatim probleme sa bazom podataka i na kraju sve ostale probleme. U ovom slučaju bi imali tri `catch` bloka. Zavisno od tipa greške, kod se automatski preusmerava na odgovarajući `catch` blok.

Blok `finally` (ako postoji) se izvršava uvek, bez obzira da li je do greške zaista došlo ili ne. Ovde upisujemo linije koda koje uvek treba da se izvrše.

Šta se događa kada se pojavi greška u vreme izvršavanja koju nismo obradili try-catch blokom?

Na formular dodajte još jedno dugme i uradite dupli klik na njega. Ovde ćemo napisati sintaksno ispravni kod koji će uvek generisati grešku u toku izvršavanja:

```
private void button2_Click (object sender, EventArgs e)
{
    int a = 10, b = 0, c;
    c = a / b;
}
```

Deklarišemo tri varijable i nakon toga namerno pravimo grešku deljenja sa nulom ($b = 0$). Pokrenite program i kliknite na dugme. Kada se greška dogodi u programu koji je pokrenut iz okruženja, on se zaustavlja i prikazuje se problematična linija se opisom greške i predlozima kako se greška može ispraviti:



Kada pokrećete kompajliranu aplikaciju (.exe ekstenzija) van razvojnog okruženja, ponašanje je naravno drugačije. Pritisnite F6 funkciski taster čime se kompajlira program. Potom nađite izvršnu verziju programa koja se nalazi u direktorijumu \bin\Debug ispod direktorijuma gde ste snimili projekat.

Poruka o grešci je prikazana na sledećoj slici:



Cilj je da korisnik vašeg programa nikada ne dobije ovakvu poruku o grešci. Iako se u dijalogu nudi opcija da se nastavi sa izvršavanjem programa (dugme Continue) u praksi nastavak rada programa posle greške obično dovodi do daljnjih grešaka i nema smisla.

Sada ćemo izmeniti kod i dodati obradu greške:

```
private void button2_Click(object sender, EventArgs e)
{
    int a = 10, b = 0, c;
    try
    {
        c = a / b;
    }
    catch
    {
        MessageBox.Show("GRESKA: deljenje sa nulom");
    }
}
```

Pokrenite aplikaciju i kliknite na drugo dugme. Sada dobijamo poruku, ali program nastavlja sa radom:



Ista poruka se dobija kada se pokrene izvršna verzija aplikacije van radnog okruženja.

U sledećem nastavku ćemo detaljno obraditi Exception objekat i alate koji služe za ispravljanje grešaka u programu.

Mala škola programiranja C# (12)

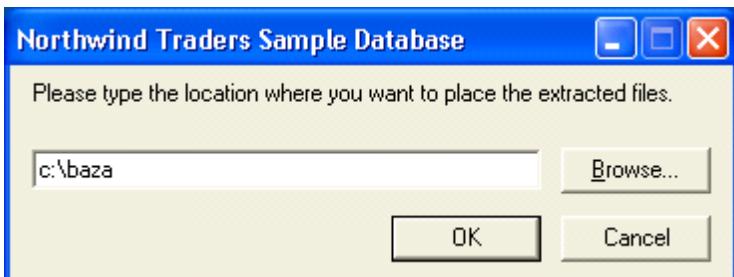
Rad sa bazama podataka

Od ovog broja započinjemo novo poglavlje u okviru C# serijala. Rad sa bazama podataka. U početku će biti objašnjene tehnologije i način pristupa i manipulacije podacima iz baze podataka. Nakon toga ćemo proučiti osnove SQL programskog jezika i naučiti da ga koristimo iz C# aplikacija. Koristićemo gotovu bazu podataka u Microsoft Access formatu koja je upravo namenjena za učenje i vežbanja koja ćemo mi raditi. Kao pripremu za naredne članke ovog serijala, neophodno je da sa Interneta preuzmete ovu bazu podataka sa lokacije

<http://www.microsoft.com/downloads/details.aspx?FamilyID=C6661372-8DBE-422B-8676-C632D66C529C&displaylang=EN>

Preuzimate datoteku Nwind.exe koja je samoraspakujuća arhiva i sadrži bazu podataka Nwind.mdb (mdb je standardna ekstenzija za Access bazu podataka). Veličina datoteke koju preuzimate je 476Kb.

Posle preuzimanja je pokrenite, odgovorite potvrđno na sva pitanja i u dijalogu gde treba da unesete lokaciju na disku gde će se raspakovati baza ukucajte c:\baza kao što je prikazano na sledećoj slici:



Jako je bitno da se baza nalazi na ovoj lokaciji jer ćemo je tako referencirati u svim narednim primerima.

Vrlo ukratko o bazi podataka

Glavni nosilac informacija u svakoj bazi podataka je tabela. Ona predstavlja dvodimenzionalnu matricu u kojoj su u redovima poređane celovite informacije. Na primer tabela u kojoj bi čuvali informacije o kupcima (eng. Customers) bi izgledala ovako:

Customers : Table			
	Customer ID	Company Name	Contact Name
▶ +	ALFKI	Alfreds Futterkiste	Maria Anders
▶ +	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
▶ +	ANTON	Antonio Moreno Taquería	Antonio Moreno
▶ +	AROUT	Around the Horn	Thomas Hardy
▶ +	BERGS	Berglunds snabbköp	Christina Berglund
▶ +	BLAUS	Blauer See Delikatessen	Hanna Moos
▶ +	BLONP	Blondel père et fils	Frédérique Citeaux
▶ +	BOLID	Bólido Comidas preparadas	Martín Sommer
▶ +	BONAP	Bon app'	Laurence Lebihan
▶ +	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln
▶ +	BSBEV	B's Beverages	Victoria Ashworth
▶ +	CACTU	Cactus Comidas para llevar	Patricia Simpson
▶ +	CENTC	Centro comercial Moctezuma	Francisco Chang
▶ +	CHOPS	Chop-suey Chinese	Yang Wang
▶ +	COMMI	Comércio Mineiro	Pedro Afonso
▶ +	CONSH	Consolidated Holdings	Elizabeth Brown

U ovom primeru svaki kupac ima atribute (nazive možete videti u zaglavlju tabele) CustomerID, CompanyName, ContactName i tako dalje. Prilikom kreiranja strukture tabele se zavisno od zahteva definišu ovi atributi – slično kao kada bi pripremali nekakav papirni upitnik. Jedan red (eng. row) predstavlja kompletну informaciju o jednom kupcu.

Jedna baza podataka može imati više različitih tabela, neretko i par stotina ili čak više hiljada u velikim sistemima. Većina tabela u bazi je u međusobnim vezama (relacijama), ali dalja rasprava prevazilazi temu ovog serijala.

Za sada treba da naučimo kako da napravimo C# aplikaciju pomoću koje može pregledati i modifikovati podatke u jednoj tabeli.

ADO.NET

ADO (ActiveX Data Object) je .NET tehnologija koja nam omogućava rad sa bazama podataka različitih formata. Predstavlja deo .NET Framework okruženja i tehnički je realizovana pomoću velikog broja gotovih klasa koje enapsuliraju sve potrebne funkcionalnosti. Neka vas ne brine predhodna rečenica, jer sa tačno četiri klase možemo obaviti gotovo sve što je potrebno.

Za početak, (zlo)upotrebićemo mogućnost Visual Studio da bez i jedne linije koda postavimo vezu ka našoj bazi i prikažemo podatke iz bilo koje tabele u njoj.

Primer u šest lakih koraka

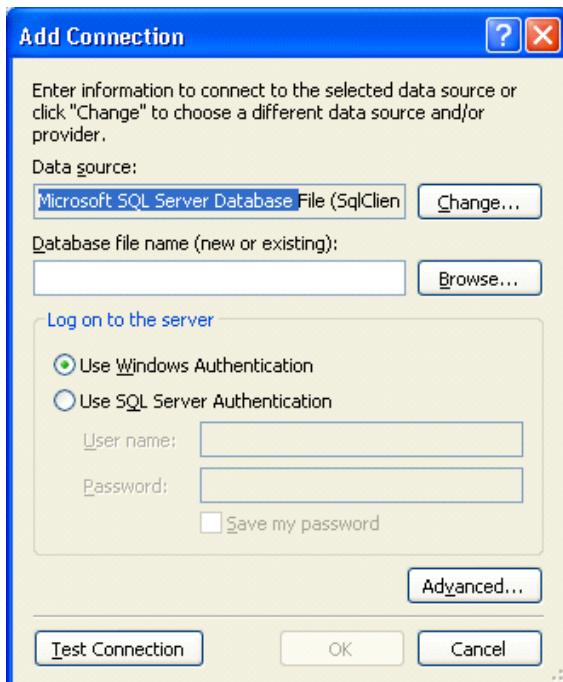
Uz prepostavku da ste preuzeли базу са Interneta и да се налази на C:\BAZA\NWIND.MDB, проćiћемо кроз следеће кораке како би постигли циљ.

Покрените Visual C# Express edition и изаберите нов проект типа Windows Application.

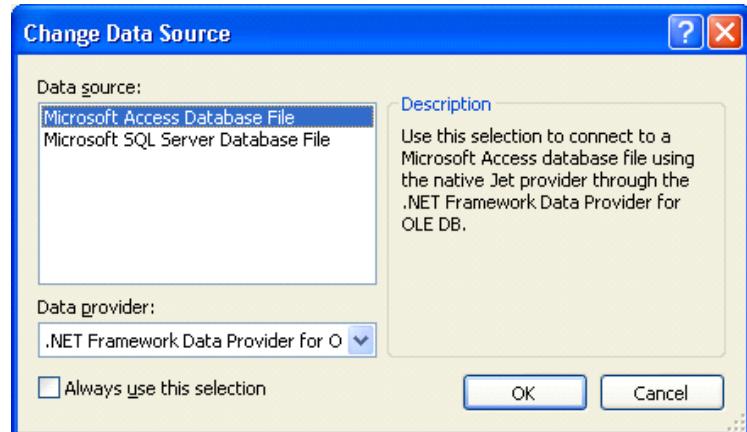
1. Pre svega, потребно је додати извор података (eng. Data Source) у пројекат. Data Source представља заправо везу ка бази података коју ћелимо да користимо. Подрžане су различити формати база, зависно од инсталираних дравера у вашем систему, али увек рачунавјте да постоји подршка за Microsoft Access и SQL Server.

Кликните на меню Data и потом изаберите ставку Add New Data Source. Овим smo покренули Data Source Configuration **čarobnjak**. На првом екрану "Choose a Data Source Type" изаберите DataBase и кликните на дугме Next.

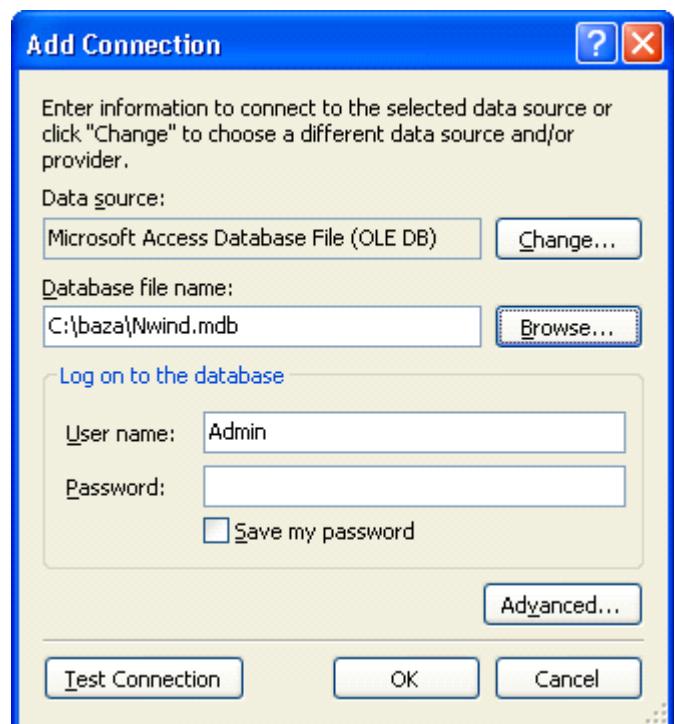
2. У овом дијалогу треба да додате везу (eng. Connection) ка бази података, као и да изаберете тип базе. Кликните на дугме New Connection. Отвара се Add Connection (додад везу) дијалог. Обратите паžњу да је иницијално постављен Microsoft Sql Server. Да би ово изменили кликните на дугме Change као што је приказано на слици.



У Change Data Source дијалогу изаберите Microsoft Access Database File, као што је приказано на слици и кликните на дугме OK.



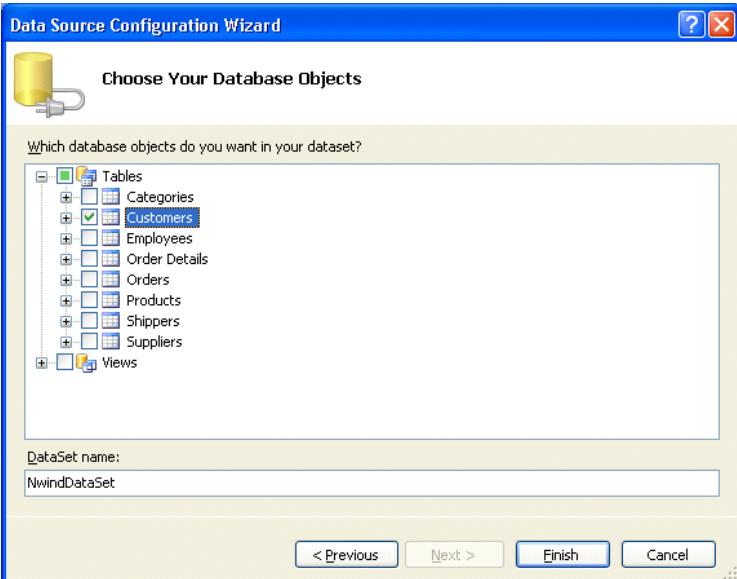
3. Сада smo се вратили на предходни дијалог где треба кликнути на дугме Browse како би коначно изабрали наšу базу података. На крају дијалог изгледа као на следећој слици. Кликните на дугме OK, чиме smo завршили дефиницију везе ка



бази Nwind.mdb и потом на дугме Next за sledeći korak. На потом постављено пitanje odgovorite sa No kako ne bi kopirali базу на локацију где је snimljen пројекат.

4. Sledeći korak "Save connection string..." nam u ovom моменту nije značajan, па ostavite како је понуђено и кликните на Next.

5. U ovom koraku "Choose your database objects" treba da izaberemo jednu ili više tabele sa kojom hoćemo da radimo. U listi se, između ostalog, nalaze sve tabele u bazi podataka. Kliknite na znak + pored Tables i izaberite tabelu Customers kao što je prikazano na slici.



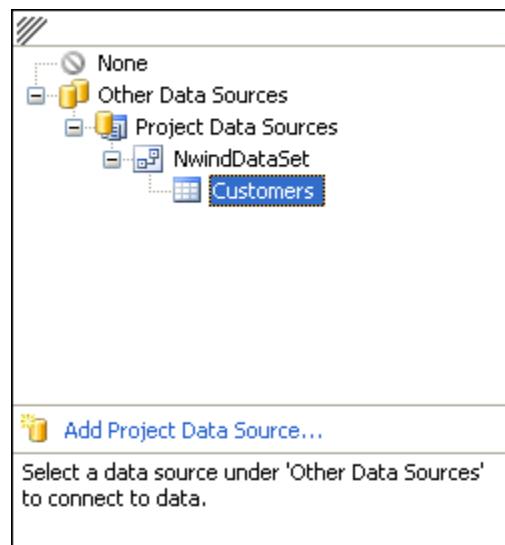
Na kraju kliknite na dugme Finish čime smo završili konfiguraciju izvora podataka.

6. Sada nam ostaje da na formi prikažemo podatke iz izvora podataka koga smo definisali. Za ovo postoji više načina jer gotovo svaka kontrola nudi mogućnost ovakvog prikaza. Kada su u pitanju tabele, najčešće se koristi kontrola DataGridView koji možete naći u paleti sa alatkama u sekciji Data. Postavite je na formu i odmah u Properties prozoru pronađite njeno svojstvo DataSource. Ovo svojstvo služi da se kontrola poveže sa izvorom podataka i na taj način u vreme izvršavanja prikaže stvarne podatke iz baze.

Otvorite padajuću listu pored svojstva DataSource i izaberite kao što je prikazano na sledećoj slici:

Primetićete da je radno okruženje automatski dodalo još neke objekte (vide se ispod forme), a takođe je DataGridView kontrola dobila zaglavljne imenima atributa tabele Customers koju smo izabrali.

I to je sve (za sada).



Pokrenite aplikaciju. Čarobnjak koga smo upravo prošli je za nas, u pozadini, napisao sav potrebnii kod. Ako ste pažljivo pratili korake trebalo bi da dobijete tabelu sa podacima iz baze:

	CustomerID	CompanyName	ContactName	ContactTi
▶	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Rep
	ANATR	Ana Trujillo Empa...	Ana Trujillo	Owner
	ANTON	Antonio Moreno ...	Antonio Moreno	Owner
	AROUT	Around the Horn	Thomas Hardy	Sales Rep
	BERGS	Berglunds snabb...	Christina Berglund	Order Adm
	BLAUS	Blauer See Delik...	Hanna Moos	Sales Rep
	BLONP	Blondel père et fils	Frédérique Citeaux	Marketing
	BOLID	Bólido Comidas p...	Martín Sommer	Owner
	BONAP	Bon app'	Laurence Lebihan	Owner
	BOTTM	Bottom-Dollar Ma...	Elizabeth Lincoln	Accountin
	BSBEV	B's Beverages	Victoria Ashworth	Sales Rep
	CACTU	Cactus Comidas ...	Patricia Simpson	Sales Age

Probajte da skrolujete po visini i širini kako bi videli sve podatke.

Kliknite na zaglavje tabele. Šta se dešava? Očigledno puno ugrađenih gotovih funkcionalnosti koje ne treba ručno programirati.

U sledećem nastavku ćemo proučiti objekte koji se u ovom primeru koriste.

Mala škola programiranja C# (13)

Rad sa bazama podataka

Da bi mogli da napišemo bilo kakav program ili web sajt koji pristupa bazi podataka neophodno je da upoznamo jezik kojim se manipuliše podacima baze podataka. U pitanju je jezik pod nazivom SQL (eng. Structured Query Language) odnosno strukturiranu jezik za zadavanje upita. Ponegde se SQL akronim interpretira kao i "Standard Query Language" odnosno standardni jezik za zadavanje upita. Kako bilo, SQL jezik je standardni jezik koji je predstavljen ranih 1970-tih i od tada je doživeo nekoliko standardizacija od kojih je poslednja bila 1999, a nakon nje još nekoliko revizija.

Dobra stvar kod standardizacije SQL jezika je što jednom naučen može biti primenjen na bazama podataka različitih proizvođača. Da ne bi sve bilo idealno, u praksi možete računati da predhodna tvrdnja nije potpuno tačna, jer svaki proizvođač uvodi nove mogućnosti koje manje ili više odstupaju od standarda.

Primarni ključ table

U prošlom nastavku je pomenuta tabela kao osnovni nosilac informacija. Osim ovoga svaka tabela poseduje jedan važan objekat – **primarni ključ table** (eng. Primary Key ili skraćeno PK). Primarni ključ predstavlja jedna ili više kolona sa ciljem da se svaki red u tabeli jedinstveno identificuje. Na primer, ako bi želeli da kreirate tabelu sa podacima studenata jednog fakulteta, najbolji kandidat za primarni ključ je broj indeksa studenta. Ako bi za primarni ključ postavili ime i prezime, pogrešili bi jer to nije jedinstvena vrednost.

Ponekad jedna vrednost nije dovoljna za jedinstvenu identifikaciju reda table. Ako bi želeli da napravimo tabelu sa podacima o studentima svih fakulteta u Srbiji, samo broj indeksa ne bi bio dovoljan. Morali bi da dodamo grad i naziv fakulteta. Ovako kreirani primarni ključ koji se sastoji od dve ili više vrednosti (u našem primeru tri), se zove kompozitni primarni ključ.

Zašto je primarni ključ važan?

Ako želimo da pogledamo podatke, izmenimo ili izbrišemo tačno određeni podatak to ne bi mogli precizno da

uradimo bez ispravno formiranog primarnog ključa.

Tehnički gledano tabela ne mora da ima primarni ključ, ali to zaista nikako nije dobra praksa.

Ponekad ne postoji ni jedna kolona u tabeli koja je jedinstvena i može da bude kandidat za primarni ključ. U tom slučaju se veštački dodaje još jedna kolona, najčešće numerička, gde baza podataka sama upisuje brojeve u rastućem nizu. Ovakva kolona, zavisno od baze podataka, se zove Autonumber, Identity i slično.

Tipovi podataka u tabeli

Zavisno od potreba, u svakoj koloni tabele se upisuju različiti tipovi podataka. Da se ne bi previše udaljavali od teme pomenućemo samo osnovne grupe:

- Tekstualni podaci – služe za slobodni unos alfanumeričkog teksta
- Celobrojni numerički podaci – unos celobrojnih pozitivnih i negativnih vrednosti
- Numerički podaci sa pokretnim zarezom
- Datumski podaci
- Specijalni tipovi – tekstualna polja velikog kapaciteta, slike, zvuk, geografski podaci i slično

Tip podatka u nekim slučajevima određuje i sintaksu SQL upita kao što ćemo videti u narednom tekstu.

Tipovi SQL upita

SQL upiti se po svojoj funkciji mogu podeliti na sledeće grupe:

- DML (eng. Data Manipulation Language), jezik za manipulaciju podataka. Ovu grupu ćemo detaljnije proučiti jer se odnosi na pregled, dodavanje, izmenu i brisanje podataka.
- DDL (eng. Data Definition Language) pomoću koga se kreiraju, menjaju i brišu objekti u bazi podataka. Pomoću njih možemo kreirati table, odrediti nazive i tipove podatka, definisati primarni ključ table i slično.
- DCL (eng. Data Control Language) – jezik za kontrolu podataka pomoću koga se određuje koji korisnik šta sme da radi sa podacima. Na primer, možete određenim korisnicima dati pravo da samo pregleđuju podatke, a nekim i da ih menjaju.

Naš cilj je upoznavanje sa jezikom za manipulaciju podataka.

Napomena:

Sledeći primeri su saglasni sa osnovnim SQL standardom. To znači da će sintaksno gledajući, moći da se primene na svakoj bazi podataka. Konkretnе табеле i подаци koji se ovde koriste se odnose na Microsoft Access Northwind bazу podataka. U predhodnom nastavku je dat Internet link sa koga možete preuzeti ovu bazu, ali za svaki slučaj još jednom: <http://www.microsoft.com/downloads/details.aspx?FamilyID=C6661372-8DBE-422B-8676-C632D66C529C&displaylang=EN>

Prikaz podataka

Prikaz podataka iz baze se dobija pomoću SELECT naredbe. Osnova sintaksna struktura SELECT naredbe je veoma jednostavna:

SELECT nazivi kolona

FROM naziv tabele

Nazivi kolona su razdvojeni zarezom, a posle FROM klauzule se upisuje ime tabele. Navedena imena kolona moraju postojati u tabeli, u suprotnom dobićemo poruku o grešci.

U Northwind bazi postoji tabela pod nazivom Customers u kojoj se nalazi lista svih kupaca sa kolonama CustomerID (šifra kupca), CompanyName (ime kompanije), City (grad) i tako dalje.

Ako bi želeli da prikažemo pomenute kolone iz tabele Customers, SELECT upit bi trebao da izgleda ovako:

```
SELECT CustomerID, CompanyName, City
FROM Customers
```

(Nisu bitna velika i mala slova – nepisano pravilo je da se SQL naredbe pišu velikim slovima)

Kada se pokrene ovakav upit, dobija se rezultat kao na slici:

Customer ID	Company Name	City
ALFKI	Alfreds Futterkiste	Berlin
ANATR	Ana Trujillo Emparedados y helados	México D.F.
ANTON	Antonio Moreno Taquería	México D.F.
AROUT	Around the Horn	London
BERGS	Berglunds snabbköp	Luleå
BLAUS	Blauer See Delikatessen	Mannheim
BLONP	Blondel père et fils	Strasbourg
BOLID	Bólido Comidas preparadas	Madrid
BONAP	Bon app'	Marseille
BOTTM	Bottom-Dollar Markets	Tsawassen
BSBEV	B's Beverages	London
CACTU	Cactus Comidas para llevar	London

Ako želite dobiti sve kolone iz tabele možete ih sve redom navesti ali postoji i skraćeni oblik – upišite zvezdicu umesto liste kolona:

```
SELECT *
FROM Customers
```

Rezultat:

Customer ID	Company Name	Contact Name	Contact Title	Address	City	Region
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin	
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 22	México D.F.	
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London	
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå	
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim	
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg	
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid	
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille	
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus	London	
CACTU	Cactus Comidas para llevar	Patricia Simpson	Sales Agent	Cerrito 333	Buenos Aires	

Ovo je i zgodan način da se brzo sazna koje kolone ima tabela.

U praksi je često potrebno da se podaci prikažu po određenom redosledu. Redosled može biti po jednoj ili više kolona. Ako bi rezultat upita želeli da složimo po gradu i to od A pa na dalje koristimo ORDER BY klauzulu:

```
SELECT CustomerID, CompanyName, City
FROM Customers
ORDER BY City ASC
```

Rezultat:

Customer ID	Company Name	City
DRACD	Drachenblut Delikatessen	Aachen
RATTC	Rattlesnake Canyon Grocer	Albuquerque
OLDWO	Old World Delicatessen	Anchorage
VAFFE	Vaffeljernet	Århus
GALED	Galería del gastrónomo	Barcelona
LILAS	LILA-Supermercado	Barquisimeto
MAGAA	Magazzini Alimentari Riunite	Bergamo
ALFKI	Alfreds Futterkiste	Berlin
CHOPS	Chop-suey Chinese	Bern
SAVEA	Save-a-lot Markets	Boise
FOLKO	Folk och fä HB	Bräcke
KOENE	Königlich Essen	Brandenburg
MAISD	Maison Dewey	Bruxelles
CACTU	Cactus Comidas para llevar	Buenos Aires

ASC je skraćeno od Ascending (rastući niz). Može se koristiti i DESC – Descending, odnosno opadajući niz gde se sortiranje obavlja od većeg ka manjem. Kod većine baza podataka ASC je podrazumevana vrednost i ne mora se eksplicitno navoditi.

Napisaćemo upit koji će sortirati po dve kolone: City i CompanyName i to City po rastućem, a CompanyName po opadajućem redosledu.

```
SELECT City, CompanyName
FROM Customers
ORDER BY City ASC, CompanyName DESC
```

City	Company Name
Lille	Folies gourmandes
Lisboa	Princesa Isabel Vinhos
Lisboa	Furia Bacalhau e Frutos do Mar
London	Seven Seas Imports
London	North/South
London	Eastern Connection
London	Consolidated Holdings
London	B's Beverages
London	Around the Horn
Luleå	Berglunds snabbköp
Lyon	Victuailles en stock
Madrid	Romero y tomillo
Madrid	FISSA Fabrica Inter. Salchichas S.
Madrid	Bólido Comidas preparadas
Mannheim	Blauer See Delikatessen
Marseille	Bon app'
México D.F.	Tortuga Restaurante
México D.F.	Pericles Comidas clásicas

U sledećem nastavku ćemo raditi postavljanje kriterijuma i vratiti se nazad na C# i implementaciju SELECT upita.

Mala škola programiranja C# (14)

Rad sa bazama podataka

U ovom nastavku nastavljamo sa prikazom mogućnosti SQL SELECT upita i nekim njegovim naprednjijim opcijama.

U praktičnom radu gotovo nikad nam ne treba prikaz svih podataka iz jedne tabele. U tom smislu, postoje dva termina: vertikalni i horizontalni filter.

Vertikalni filter znači da ne prikazujemo sve kolone iz tabele, već samo one koje nam zaista trebaju. Na primer, želite videti samo šifru i naziv kompanije i nikakve druge podatke. Ovo se implementira jednostavno tako što u SELECT listi navedete samo potrebna polja.

Horizontalni filter znači da ne želite da vidite sve slogove (redove) iz tabele. Na primer, želite da vidite samo kupce za zadati grad. Ovo se postiže pomoću WHERE naredbe u kojoj se definiše jedan ili više kriterijuma koji moraju biti ispunjeni da bi se slog prikazao. WHERE naredba (još se zove klauzula) je veoma slična if naredbi u programskim jezicima. Moguće je zadati više kriterijuma koji se međusobno povezuju OR i AND operatorima.

Ako želimo da vidimo sve kupce iz grada London, upit bi izgledao ovako:

```
SELECT CustomerID, CompanyName, City
FROM Customers
WHERE City = 'London'
```

Sintaksa je jednostavna: posle WHERE klauzule se navodi ime polja (City), zatim operator (=) i na kraju vrednost ('London'). Obratite pažnju da se vrednost London nalazi između jednostrukih znakova za navođenje. Pošto je polje City tekstualnog tipa, sintaksno, kriterijum mora ovako da se piše. Ovo usvojite kao pravilo koje se poštuje u svim bazama podataka.

Rezultat predhodnog upita:

Customer ID	Company Name	City
AROUT	Around the Horn	London
BSBEV	B's Beverages	London
CONSH	Consolidated Holdings	London
EASTC	Eastern Connection	London
NORTS	North/South	London
SEVES	Seven Seas Imports	London
*		

Record: 1 of 6 No Filter Search

Ako na primer, želimo da prikažemo kupce iz Londona i Berlina onda se radi o složenom uslovu sa dva kriterijuma. Čitate ovako: prikaži kupce koji su iz Londona ili Berlina – znači radi se o OR (eng. ili) operatoru.

```
SELECT CustomerID, CompanyName, City
FROM Customers
WHERE City = 'London' OR City = 'Berlin'
```

Rezultat upita:

Customer ID	Company Name	City
ALFKI	Alfreds Futterkiste	Berlin
AROUT	Around the Horn	London
BSBEV	B's Beverages	London
CONSH	Consolidated Holdings	London
EASTC	Eastern Connection	London
NORTS	North/South	London
SEVES	Seven Seas Imports	London
*		

Record: 1 of 7 No Filter Search

Pitanje za čitaoca: ako bi u predhodnom primeru operator OR zamenili sa operatorom AND, kakav bi rezultat doobili?

Odgovor: ne bi dobili ni jedan slog jer (logično) ni jedan kupac ne može istovremeno da bude i iz Londona i iz Berlina :)

Često se u kombinuju WHERE i ORDER BY klauzula koja služi za sortiranje i koju smo opisali u predhodnom nastavku. U tom slučaju neophodno je ispoštovati redosled klauzula u SELECT upitu:

```
SELECT ...
FROM ...
WHERE ...
ORDER BY ...
```

Na primer:

```
SELECT ContactName, CustomerID, CompanyName, City
FROM Customers
WHERE City = 'London' OR City = 'Berlin'
ORDER BY ContactName
```

Rezultat upita:

Contact Name	Customer I	Company Name	City
Ann Devon	EASTC	Eastern Connection	London
Elizabeth Brown	CONSH	Consolidated Holdings	London
Hari Kumar	SEVES	Seven Seas Imports	London
Maria Anders	ALFKI	Alfreds Futterkiste	Berlin
Simon Crowther	NORTS	North/South	London
Thomas Hardy	AROUT	Around the Horn	London
Victoria Ashworth	BSBEV	B's Beverages	London

Možete kreirati proizvoljno složene kriterijume koristeći više OR i/ili AND operatora, ali vodite računa o prioritetu: AND ima viši prioritet od OR. Kako se tačno tumači sledeći kriterijum:

...

```
WHERE City = 'London' AND CustomerID = 'ALFKI' OR
Country = 'Italy'
```

Pošto AND ima prioritet, uslov čitamo na sledeći način:

...

```
WHERE (City = 'London' AND CustomerID = 'ALFKI') OR
Country = 'Italy'
```

Postavljanjem uslova u zagrade možete forsirati potrebnii prioritet po pravilima bulove algebre.

Ako želite da prikažete sve podatke osim nekog po zadatom kriterijumu, možete koristiti operator negacije NOT. Na primer želimo videti sve kupce osim onih iz Londona:

```
SELECT CustomerID, CompanyName, City
FROM Customers
WHERE NOT City = 'London'
```

Postoji još jedna zgodna skraćenica u pisanju. Ako za isto polje postoji više kriterijuma vezanih sa operatorom OR, može se koristiti IN operator. Na primer, ako želimo da

vidimo sve kupce iz gradova London, Berlin, Madrid i Pariz kriterijum bi se skraćeno napisao na sledeći način:

```
SELECT ContactName, CustomerID, CompanyName, City
FROM Customers
WHERE City IN ('London', 'Berlin', 'Madrid', 'Paris')
```

Rezultat upita:

Contact Name	Customer I	Company Name	City
Maria Anders	ALFKI	Alfreds Futterkiste	Berlin
Thomas Hardy	AROUT	Around the Horn	London
Martín Sommer	BOLID	Bólido Comidas preparadas	Madrid
Victoria Ashworth	BSBEV	B's Beverages	London
Elizabeth Brown	CONSH	Consolidated Holdings	London
Ann Devon	EASTC	Eastern Connection	London
Diego Roel	FISSA	FISSA Fabrica Inter. Salchichas S.A.	Madrid
Simon Crowther	NORTS	North/South	London
Marie Bertrand	PARIS	Paris spécialités	Paris
Alejandra Camino	ROMEY	Romero y tomillo	Madrid
Hari Kumar	SEVES	Seven Seas Imports	London
Dominique Perrier	SPEC'D	Spécialités du monde	Paris

Kada su polja u kriterijumu numeričkog tipa, uslov se piše isto kao i do sada ali ovog puta vrednost kriterijuma ne sme da se stavlja u jednostrukne znakove navoda. Na primer želimo videti sve proizvode čija je cena (polje UnitPrice) veća od 50 i da rezultat sortiramo po istom polju.

```
SELECT ProductID, ProductName, UnitPrice
FROM Products
WHERE UnitPrice > 50
ORDER BY UnitPrice
```

Rezultat upita:

Slično IN operatoru postoji još jedan skraćeni način pisanja

Product I	Product Name	Unit Price
51	Manjimup Dried Apples	53
59	Raclette Courdavault	55
18	Carnarvon Tigers	62.5
20	Sir Rodney's Marmalade	81
9	Mishi Kobe Niku	97
29	Thüringer Rostbratwurst	123.79
38	Côte de Blaye	263.5

kriterijuma. Ako želimo da vidimo sve proizvode čije su cene između 50 i 100 (uključujući i ove vrednosti) regularno bi napisali:

```
SELECT ProductID, ProductName, UnitPrice
FROM Products
WHERE UnitPrice >= 50 AND UnitPrice <= 100
```

Operator \geq označava "veće ili jednako".

Operator \leq znači "manje ili jednako".

Kriterijum možemo skratiti i što je možda važnije, učiniti čitljivijim pomoću BETWEEN operatora:

```
SELECT ProductID, ProductName, UnitPrice
FROM Products
WHERE UnitPrice BETWEEN 50 AND 100
```

Rezultat upita:

The screenshot shows the Microsoft Query application window titled 'Query1'. It displays a table with three columns: 'Product ID', 'Product Name', and 'Unit Price'. The data rows are: 9 Mishi Kobe Niku (Unit Price 97), 18 Carnarvon Tigers (Unit Price 62.5), 20 Sir Rodney's Marmalade (Unit Price 81), 51 Manjimup Dried Apples (Unit Price 53), and 59 Raclette Courdavault (Unit Price 55). The status bar at the bottom indicates 'Record: 1 of 5'.

Product ID	Product Name	Unit Price
9	Mishi Kobe Niku	97
18	Carnarvon Tigers	62.5
20	Sir Rodney's Marmalade	81
51	Manjimup Dried Apples	53
59	Raclette Courdavault	55

Vodite računa da korišćenje BETWEEN operatora uvek uključuje i gornju i donju granicu.

Korišćenje horizontalnog i vertikalnog filtera je veoma bitno jer značajno smanjuje opterećenje servera baze podataka i mrežne infrastrukture. Imajte u vidu da u praksi nikad nije potrebno gledati sve podatke. Pažljivo napisani upiti, gde se prikazuje samo ono što je stvarno potrebno (i po horizontali i po vertikali), mogu značajno da poboljšaju ukupne performanse sistema.

Mala škola programiranja C# (15)

Rad sa bazama podataka

U predhodnom nastavku smo se upoznali sa osnovama SELECT naredbe SQL upita. Ovo nam daje dovoljnu osnovu da dalje naučimo kako napraviti aplikaciju koja bazi podataka zadaje SELECT upit i prikazuje rezultat njegovog izvršavanja. Za primer ćemo koristiti već pomenutu Nwind bazu podataka u Microsoft Access formatu. Preduslov je da na disku C: imate direktorijum baza i u njemu datoteku Nwind.mdb. Ovo je važno, jer se prilikom obraćanja bazi referenciramo na njenu putanju – u našem slučaju C:\baza\Nwind.mdb. Prijemite se za malo teorije koja nam je neophodna kako bi u sledećem nastavku napravili odgovarajući primer.

Imenovani prostori i upravljački programi

Prilikom rada sa bazama podataka, C# i generalno .NET radno okruženje nude veliki broj klasa koje su raspoređene u dva imenovana prostora (Eng: name spaces):

- System.Data.SqlClient
- System.Data.OleDb

Oba se nalaze hijerarhijski ispod System.Data prostora imena. Šablon "Windows Application" inicijalno već ima referencu na System.Data, tako da nema potrebe praviti dodatne reference u projektu. Međutim, kao i do sada, zgodno je iskoristiti using direktivu kako bi jednostavno skratili pisanje prilikom kodiranja.

U imenovanom prostoru System.Data.SqlClient se nalazi set klasa koja služi za rad sa bazom podataka Microsoft SQL Server. Imenovani prostor System.Data.OleDb sadrži klase kojima se radi sa ostalim formatima baza podataka. Termin koji treba razumeti je "drajver za bazu podataka". Možemo napraviti poređenje sa drajverom (pogonskim programom) za grafičku karticu računara. Da bi kartica radila u bilo kom operativnom sistemu, neophodno je instalirati odgovarajući drajver. U pitanju je softver koji predstavlja vezu između operativnog sistema i fizičkog hardvera u računaru.

Slično je i sa bazama podataka. Drajver za bazu podataka omogućava sve funkcionalnosti za zadati format i na taj način (na sreću) skriva od programera kompleksnost prilikom rada sa različitim formatima baza na tržištu. Ovo je koncept koje se naziva enkapsulacija. Na dalje se stvari

malo komplikuju jer, istorijski gledano, postoji više tehnologija od kojih je svaka donela svoj set drajvera. Da ne ulazimo previše u priču, OleDb je tehnologija koja se koristi za pristup bazama kao što su Access, FoxPro, stari DBF i slični. Takođe je moguće sa Web sajtova proizvođača baza podataka preuzeti različite tipove drajvera za njihov format. Na primer Oracle i MySql omogućavaju besplatno preuzimanje drajvera bilo u OleDb ili .NET tehnologiji. Odlična stvar kod ovakvog koncepta je što se sistem rada sa bazama međusobno minimalno razlikuje. Uvek su u pitanju iste klase sa istim ili sličnim funkcionalnostima. U praksi ovo znači da nije komplikovano napisati aplikaciju koja će podjednako dobro raditi sa različitim formatima baze podataka – rezultat je proširena vrednost aplikacije jer, na primer, korisnik može izabrati koju bazu želi da koristi, a programer može da napiše aplikaciju koja radi sa postojećom bazom i podacima.

Pošto koristimo Access bazu podataka, treba koristiti OleDb drajvere, odnosno klase u System.Data.OleDb imenovanom prostoru.

Klase za rad sa bazama podataka

Govorili smo o dva imenovana prostora koji imaju isti set klasa. Razlikuju se u prefiksima naziva. Na primer imamo SqlConnection (System.Data.SqlClient) i OleDbConnection (System.Data.OleDb imenovani prostor). Ovo imajte u vidu jer u narednom tekstu radimo sa OleDb* objektima zbog Access baze podataka.

Konekcija

U svakom slučaju, bez obzira šta radili sa bazom podataka, neophodno je prvo napraviti vezu – konekciju (Eng: Connection) ka njoj. Ovu funkcionalnost nam daje klasa OleDbConnection. Osnovno svojstvo ove klase koje i određuje parametre konekcije je svojstvoConnectionString. Kao što se iz imena da prepostaviti svojstvo je tipa string. Setite se nastavka #12 ovog serijala kada smo pomoću čarobnjaka indirektno postavljali ovo svojstvo. Potrebno je podesiti string tako da se sastoji od parova: naziv parametra1=vrednost parametra1; naziv parametra2=vrednost parametra2;... i tako redom. Očigledno, separator je znak :

Parametri i njihovo značenje je sledeće:

Provider – označava ime potrebnog drajvera baze podataka kako je registrovan u sistemu. Za Access bazu ime drajvera je uvek Microsoft.Jet.OLEDB.4.0

Kada ovo sklopimo dobijamo:

"Provider= Microsoft.Jet.OLEDB.4.0;"

Data Source – kada je Access baza u pitanju, ovaj parametar označava putanju i ime datoteke koja predstavlja bazu. U našem slučaju baza se nalazi na putanji C:\baza\Nwind.mdb

Sada ConnectionString izgleda ovako:

"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\baza\Nwind.mdb"

U osnovnoj varijanti ovo je dovoljno. Međutim, ako je baza zaštićena korisničkim imenom i šifrom, neophodno je i to navesti. Na primer korisnik naše baze je Admin sa šifrom 123 (u praksi izbegavajte ovako "jake" šifre). Da bi ovo postigli koristimo još dva parametra: User id i Password. Finalna verzija stringa konekcije bi mogla da izgleda ovako:

"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\baza\Nwind.mdb;User id=Admin;Password=123"

Kada smo postavili ConnectionString svojstvo potrebno je eksplicitno otvoriti konekciju ka bazi. Za ovo se koristi jednostavna metoda Open() bez argumenata. U slučaju da je baza iz nekog razloga nedostupna, oštećena ili postoji problem u mreži, metoda Open() će generisati grešku u vreme izvršavanja.

Kada završimo posao, potrebno je zatvoriti konekciju sa bazom. Ovo tehnički nije neophodno, ali obavezno to uradite kako bi sačuvali resurse (broj istovremenih konekcija) – baza, mreža i na kraju korisnici vaše aplikacije će vam biti zahvalni ako poštujete ovo pravilo.

NAPOMENA

1. Strogo obratite pažnju na parametre Data Source i User id. Morate ih pisati sa jednim razmakom između reči, u suprotnom dobijate poruku o grešci.

2. Kada postavljate ovo svojstvo OleDbConnection objekta, treba voditi računa o specifičnosti rada sa stringovima u jeziku C#. Znak \ (obrnuta kosa crta) u stringu ima specijalno značenje – takozvani Escape ili kontrolni karakter posle koga se očekuje najčešće specijalni znak za formiranje. Zbog ovoga, ako na primer napišete:

Obj.ConnectionString =

"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\baza\Nwind.mdb"

Odmah dobijate sintaksnu grešku. Postoje dva rešenja.

Možete napisati ispred stringa znak @ koji govori kompjleru da znak \ tretira kao i svaki drugi:

Obj.ConnectionString =

@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\baza\Nwind.mdb"

Ili možete posle svakog znaka \ jednostavno dopisati još jedan:

Obj.ConnectionString =

"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\baza\\Nwind.mdb"

Oba načina su ispravna – izaberite koji vam se više svidja.

3. Ako želite da u mreži više korisnika pristupa istoj Access bazi podataka, bazu je potrebno postaviti u deljeni direktorijum na jednom računaru. Sa ostalih računara se bazi pristupa tako što se za Data Source postavi puna mrežna putanja do datoteke. Na primer ako na računaru čije je mrežno ime Server, u direktorijum Podaci smesti baza Nwind.mdb i potom podeli (Eng: share) ovaj direktorijum, puna putanja do baze je: \\Server\Podaci\Nwind.mdb a ConnectionString je:

Obj.ConnectionString=@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=\\Server\Podaci\Nwind.mdb "

Takođe morate svim korisnicima data pravo čitanja i pisanja (Eng: read/write) nad datotekom Nwind.mdb

Komanda

Objekat pomoću kojeg zadajemo SQL upit bazi podataka je OleDbCommand. Komanda se uvek izvršava preko već definisane konekcije. Može predstavljati različite stvari koje su izvan opsega ovog serijala. Mi ćemo se ograničiti na jednostavno slanje osnovnih SQL upita i prikaza rezultata izvršavanja.

Svojstva komande i njihovo značenje su sledeće:

Svojstvo Connection definiše po kojoj konekciji će komanda biti izvršena. Svojstvo je objektnog tipa i to OleDbConnection.

Svojstvo CommandText je string tipa i pomoću njega postavljamo potrebni upit.

U slučaju da želimo da izvršimo SQL SELECT upit koji smo obradili u predhodnom nastavku, neophodna su nam još dva objekta.

DataSet

DataSet predstavlja memorijski prostor u kome se čuva rezultat SELECT upita. Možemo ga tretirati kao matricu koja se sastoji od redova i kolona. Tačnije rečeno, DataSet može biti i niz matrica jer kroz jednu komandu možemo poslati više SELECT upita. Tada svaki SELECT pravi svoju matricu koja je u ovom kontekstu predstavljena objektom Table.

Obратite pažnju da DataSet nema prefiks Sql niti OleDb. DataSet se koristi kao univerzalni skladišni prostor koji čak ne mora biti ni na koji način povezan sa bazama podataka. Poseduje puno svojstava i metoda, ali o njima kasnije.

OleDbDataAdapter

Ovaj objekat predstavlja sponu između komande i objekta DataSet. OleDbDataAdapter je zadužen za pokretanje komande, prosleđivanje SELECT upita preko konekcije i na kraju popunjavanje DataSet objekta podacima koji su rezultat SELECT upita.

Svojstvo koje definiše komandu koju treba izvršiti je SelectCommand. Ovo svojstvo je tipa OleDbCommand.

Metoda koja popunjava DataSet podacima se zove Fill, a njen ulazni argument je objekat tipa DataSet.

Kada sve ovo uradimo kako treba, rezultat je DataSet popunjen podacima koga koristimo za prikaz na ekranu ili štampanje izveštaja.

Čitaocu koji se prvi put susreće sa ovim objektima sve verovatno izgleda (pre)komplikovano na prvi pogled. Međutim kreatori ove tehnologije su imali u vidu mogućnosti rada sa praktično svim postojećim formatima baza podataka koji su trenutno na tržištu ili više nisu, ali su od značaja.

U sledećem nastavku ćemo ovu teoriju prevesti u praktični primer i tada će sve doći na svoje mesto. Posle malo prakse biće očigledno koliko je ovaj koncept fleksibilan i prilagodljiv u realnom radu.

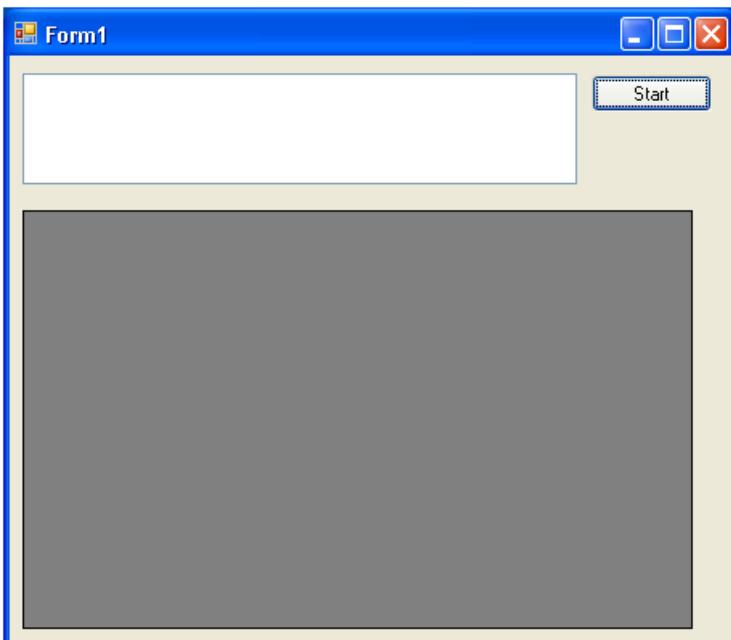
Mala škola programiranja C# (16)

Rad sa bazama podataka

U prethodnom nastavku serijala smo se pozabavili teoretskim delom pristupa podacima pomoću ADO.NET tehnologije. U ovom nastavku ćemo naučeno iskoristiti u primeru i tako teoriju preneti u praktični primer.

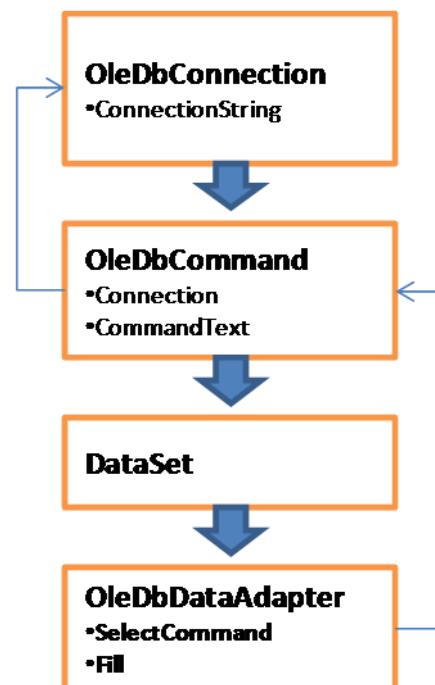
Želimo da napravimo C# aplikaciju u kojoj korisnik može da unese SQL SELECT upit, potom klikne na dugme i na kraju vidi rezultat – redove i kolone ovog upita. Da bi ovo uradili potrebno je kao i do sada da u direktorijumu C:\baza imate kopiju Access baze podataka Nwind.mdb. Ovu bazu smo koristili ranije i preporučujem čitaocima da pročitaju prethodne nastavke u vezi ove baze i SQL SELECT upita.

Korisnički interfejs aplikacije je jednostavan i njen krajnji izgled je prikazan na sledećoj slici:



Osim poznatih kontrola (TextBox i Button) upoznaćemo još jednu – DataGridView. U pitanju je veoma moćna kontrola koja je sposobna da prikaže sadržaj tabele u DataSet objektu. Da se podsetimo, DataSet objekat čuva jedan ili više Table objekata u memoriji. Svaki Table objekat je matrica sastavljena od redova i kolona i čuva rezultat jednog SELECT upita. U ovom primeru ćemo imati samo jedan Table objekat u DataSet-u koji će čuvati rezultat SELECT upita upisanog u TextBox kontrolu. DataGridView kontrolu ćemo iskoristiti za prikaz sadržaja ove tabele.

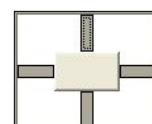
Svaki put kada želite da zadate SELECT upit bazi podataka procedure je ista. Na sledećem dijagramu je prikazan redosled kreiranja objekta, njihove veze i ključna svojstva svakog od njih:



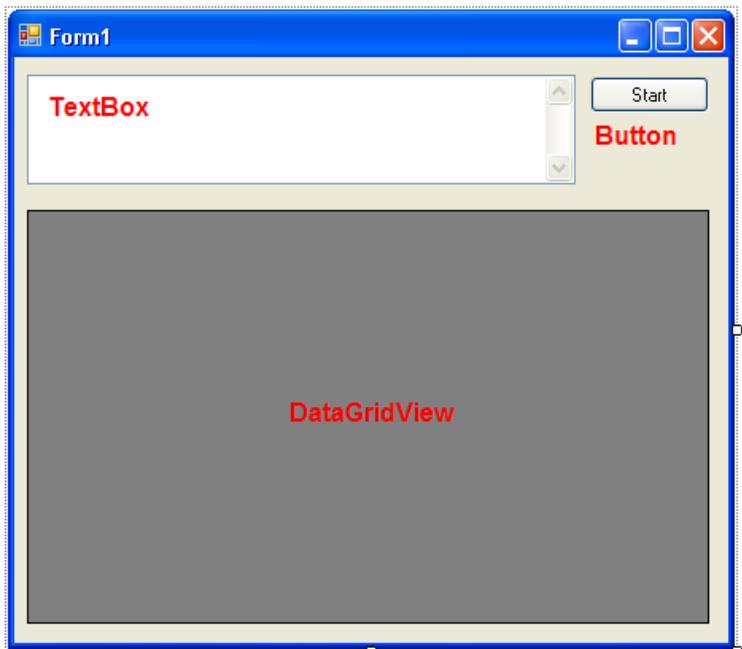
Potrebno je ispoštovati redosled rada sa objektima i kao rezultat poslednje – Fill metode dobijamo popunjeni DataSet objekat koga možemo prikazati korisniku pomoću DataGridView ili neke druge kontrole.

Pokrenite Microsoft C# Express Edition, kreirajte nov projekt tipa Windows Application i na inicijalnu formu postavite sledeće kontrole:

- TextBox – postavite svojstvo MultiLine na True, kako bi korisnik mogao da otkuca SQL SELECT upit u više redova. U skladu sa ovim postavite i svojstvo ScrollBars na Vertical. Inicijalno ime kontrole ćemo zadržati – textBox1
- Button – postavite svojstvo Text na Start i takođe ostavite inicijalno ime kontrole na button1
- DataGridView – postavite je tako da zauzima veći deo forme. Takođe želimo da kontrola prati izmene dimenzija forme što se postiže postavljanjem Anchor svojstva. Kontrolu treba privezati (eng. Anchor) na sve četiri strane, čime se automatski održava isto rastojanje od ivica forme.



Na kraju naša forma treba da izgleda ovako:



Klikom na dugme "Start" se izvršava sve, od postavljanja konekcije do vezivanja podataka na DataGridView kontrolu.

Uradite dupli klik na dugme i pre nego što krenemo sa pisanjem koda, zgodno je (ne i neophodno) dodati jednu using direktivu kako bi sebi skratili pisanje. Na vrhu prozora za pisanje koda ćete uočiti nekoliko postojećih using direktiva, a potrebno je dodati još jednu:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
```

Vraćamo se na button1_Click događaj i po redu kako je prikazano na prethodnom dijagramu pišemo kod:

Konekcija:

```
OleDbConnection Konekcija = new OleDbConnection();
Konekcija.ConnectionString =
@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source =
C:\baza\Nwind.mdb";
```

Ovde pažljivo upišite vrednost svojstva ConnectionString. Velika i mala slova nisu bitna, ali razmak u parametru Data Source jeste bitan.

Komanda:

```
OleDbCommand Komanda = new OleDbCommand();
Komanda.Connection = Konekcija;
Komanda.CommandText = textBox1.Text;
```

Tekst ukucan u kontrolu textBox1 koristimo za CommandText svojstvo komande, tako da to mora biti validan SQL SELECT upit.

DataSet:

```
DataSet Ds = new DataSet();
```

Ovde radimo samo instanciranje objekta Ds na osnovu DataSet klase.

DataAdapter:

```
OleDbDataAdapter Da = new OleDbDataAdapter();
Da.SelectCommand = Komanda;
```

Instanciranje Da objekta na osnovu OleDbDataAdapter klase.

Punjjenje DataSet-a:

```
Da.Fill(Ds);
```

Koristimo Fill metodu OleDbDataAdapter klase za "punjenje" DataSet-a.

Ove metoda zapravo pokreće SelectCommand komandu, koja dalje otvara konekciju i bazi prosleđuje upit. Rezultat ovog upita se prosleđuje DataSet-u koji pravi jednu ili više tabela.

Prikaz sadržaja prve tabele DataSet-a u DataGridView1 kontroli:

```
DataGridView1.DataSource = Ds.Tables[0];
```

Svojstvo DataSource, kao što možete prepostaviti iz naziva, pokazuje koji je izvor podataka koje treba prikazati u kontroli. U ovom primeru, izvor podataka je prvi (i jedina) tabela u DataSet-u koji ima index 0 (Ds.Tables[0]).

Kompletan kod za click događaj dugmeta:

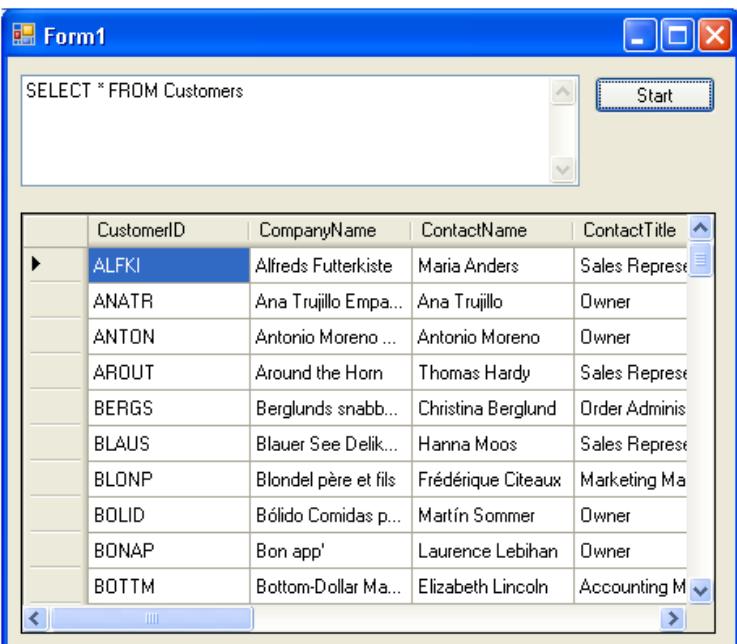
```
OleDbConnection Konekcija = new OleDbConnection();
Konekcija.ConnectionString =
@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\baza\Nwind.mdb";
OleDbCommand Komanda = new OleDbCommand();
Komanda.Connection = Konekcija;
Komanda.CommandText = textBox1.Text;
DataSet Ds = new DataSet();
OleDbDataAdapter Da = new OleDbDataAdapter();
Da.SelectCommand = Komanda;
Da.Fill(Ds);
DataGridView1.DataSource = Ds.Tables[0];
```

Vreme je da pokrenemo primer. U textBox1 ukucajte sledeći upit:

```
SELECT * FROM Customers
(Customers je tabela u nwind.mdb bazi)
```

I kliknite na dugme Start.

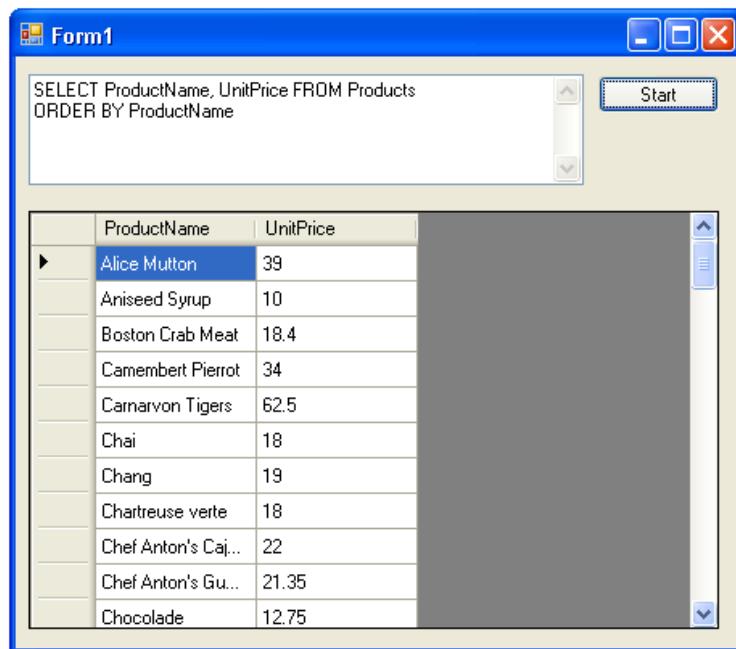
Ako ste sve tačno ukucali, treba da dobijete sledeće:



Primetićete da se DataGridView kontrola automatski podesila (redovi i kolone), zavisno od podataka koje treba da prikaže (DataSource svojstvo).

Pokušajte sledeći upit:

```
SELECT ProductName, UnitPrice FROM Products ORDER BY ProductName
(Prikaži naziv i cenu proizvoda iz tabele Products i sortiraj po nazivu proizvoda).
```



Za dalje vežbanje:

Otvorite bazu podataka Nwind.mdb u Microsoft Access aplikaciji i analizirajte njen sadržaj. Ako ne posedujete Access aplikaciju – mala pomoć: neke od tabela sa kojima možete da vežbate su: Customers, Orders, Products, Categories, Suppliers i druge.

U sledećem nastavku ćemo se detaljnije pozabaviti tehnologijom za vezivanje podataka (eng. Data Binding).

Mala škola programiranja C# (17)

Data Binding

U ovom nastavku ćemo obraditi vezivanje sa podacima (Eng. Data binding).

U prošlom nastavku smo zapravo radili data binding kada smo postavljali DataSource svojstvo kontrole DataGridView. Sličnu stvar možemo uraditi i sa gotovo svakom kontrolom korisničkog interfejsa. Vezivanje podataka znači da se u nekoj kontroli automatski prikazuju podaci iz baze, najčešće kao rezultat SELECT upita. Pošto SELECT upit može da obuhvati podatke iz jedne ili više tabela, ovo predstavlja veoma elegantan način za prikaz podataka korisniku, uz minimum programiranja. Kontrole se povezuju na potrebnu tabelu u okviru DataSet objekta.

Princip rada je identičan kao što je demonstrirano u predhodnom primeru, uz dodatak par linija koda koje su odgovorne za povezivanje podataka. Postoje dve varijante koje možemo koristiti i čiji izbor zavisi od mogućnosti same kontrole. Neke kontrole su u stanju da tabelarno prikažu podatke, kao što je DataGridView. U ovom slučaju potrebno je podesiti samo DataSource svojstvo i to je sve. Ovaj način vezivanja se zove simple data binding.

Neke druge kontrole kao što je TextBox, Label i slične očigledno ne mogu prikazati redove i kolone podataka, već mogu da prikažu samo vrednost određene kolone (polja iz tabele), određenog sloga (reda u tabeli). Ovo nas dovodi do pojma Current record (aktivni slog-zapis) čije vrednosti kolone želimo da prikažemo u kontroli.

Kada "napunimo" DataSet podacima, inicijalno aktivni slog je prvi (pod uslovom da uopšte postoje slogovi). U toku rada korisnik može da vrši navigaciju po slogovima i na taj način menja aktivni slog koji se prikazuje u ovako vezanim kontrolama.

Svaka kontrola koja podržava vezivanje podataka poseduje kolekciju DataBindings koja može da sadrži više vezivanja podataka za istu kontrolu, a u praksi se koristi najčešće samo jedna. Važno je napomenuti da možemo odrediti na koje svojstvo kontrole želimo da vežemo vrednost kolone aktivnog sloga. Na primer ako želimo da podatke

prikažemo u TextBox kontroli, to činimo preko njenog svojstva Text. Ako u podacima imamo kolonu koja može imati samo vrednosti tačno/netačno (Eng. true/false) logično je koristiti CheckBox kontrolu i vezati podatke na njen svojstvo Checked.

Sintaksa za povezivanje svojstva kontrole sa kolonom tabele iz DataSet objekta je raznolika, ali najčešće se koristi sledeća:

(predpostavka je da na formi imamo TextBox kontrolu u kojoj želimo da prikažemo vrednost kolone "CompanyName" iz već napunjenoj DataSet objekta)

```
TextBox1.DataBindings.Add ("Text", Ds.Tables[0],  
"CompanyName");
```

U ovoj liniji koda za kontrolu TextBox1 dodajemo novo vezivanje tako što za svojstvo Text iz prve tabele DataSet-a pod nazivom Ds, vezujemo kolonu "CompanyName". Vodite računa da ne vezujete dva ili više puta podatke za isto svojstvo kontrole jer ćete tada dobiti poruku o grešci.

Primer

Da bi ovo ilustrovali napravićemo jednostavan primer za simple i complex data binds. Kao i ranije koristimo Microsoft Access bazu podataka. Pokrenite Microsoft C# 2005 Express Edition i kreirajte novi projekt tipa Windows Application.

Sve što treba ćemo uraditi u Form_Load događaju kako bi se odmah po otvaranju forme prikazali i podaci.

Kao i u prošlom nastavku koristimo objekte:

- OleDbConnection
- OleDbCommand
- DataSet
- OleDbDataAdapter

Za prikaz podataka koristićemo NWind.mdb bazu i sledeće kolone iz tabele Products:

- ProductName (naziv proizvoda)
- UnitPrice (cena proizvoda)
- Discontinued (da li je proizvod dostupan)

SELECT upit koji ćemo poslati bazi je:

```
SELECT ProductName, UnitPrice, Discontinued
FROM Products
ORDER BY ProductName
```

Uradite dupli klik na formu kako bi dobili Form1_Load događaj. Setite se da dodate using direktivu kako bi skrati li pisanje koda:
using System.Data.OleDb;

U Form1_Load funkciji upišite sledeće linije koda:

```
private void Form1_Load(object sender, EventArgs e)
{
    OleDbConnection Konekcija = new OleDbConnection();
    Konekcija.ConnectionString =
        @"Provider=Microsoft.Jet.OLEDB.4.0;
        Data Source=C:\baza\Nwind.mdb";
    OleDbCommand Komanda = new OleDbCommand();
    Komanda.Connection = Konekcija;
    Komanda.CommandText =
        "SELECT ProductName, UnitPrice, Discontinued
        FROM Products
        ORDER BY ProductName";
    DataSet Ds = new DataSet();
    OleDbDataAdapter Da = new OleDbDataAdapter();
    Da.SelectCommand = Komanda;
    Da.Fill(Ds);
}
```

Ovde smo pripremili teren za simple i complex povezivanje podataka.

Sada na formu treba postaviti elemente korisničkog interfejsa. Na formu postavite redom:

- Jednu DataGridView kontrolu
- Dve Label kontrole
- Dve TextBox kontrole
- Jednu CheckBox kontrolu

Svojstva Text za Label i CheckBox kontrole postavite kao što je prikazano na sledećoj slici:



Sada želimo da podatke prikažemo u DataGridView1 kontroli koja omogućava simple data binding.

U nastavku koda je dovoljno dopisati samo jednu liniju koda:

```
dataGridView1.DataSource = Ds.Tables[0];
```

U dve TextBox kontrole želimo da prikažemo vrednosti kolona "ProductName" i "UnitPrice". Ovo je complex data binding. U nastavku upišite sledeće dve linije koda:

```
textBox1.DataBindings.Add("Text", Ds.Tables[0],
    "ProductName");
textBox2.DataBindings.Add("Text", Ds.Tables[0],
    "UnitPrice");
```

Na kraju želimo da vrednost kolone "Discontinued" prikažemo u CheckBox kontroli. Vrednosti u ovoj koloni mogu biti samo true ili false, zbog čega smo i izabrali CheckBox za njen prikaz. Dopišite još jednu liniju koda:

```
checkBox1.DataBindings.Add("Checked", Ds.Tables[0],
    "Discontinued");
```

Obratite pažnju da ovde ne vezujemo svojstvo "Text" već svojstvo "Checked".

Sada pokrenite primer. Ako ste sve ispravno upisali treba da dobijete rezultat kao na sledećoj slici:

	ProductName	UnitPrice	Discontinued
►	Chef Anton's Gumbo ...	21.35	<input checked="" type="checkbox"/>
	Chocolade	12.75	<input type="checkbox"/>
	Côte de Blaye	263.5	<input type="checkbox"/>
	Escargots de Bourgog...	13.25	<input type="checkbox"/>
	Filo Mix	7	<input type="checkbox"/>
	Fløtemysost	21.5	<input type="checkbox"/>
	Geitost	2.5	<input type="checkbox"/>

Naziv proizvoda: Chef Anton's Gumbo Mix

Cena: 21.35

Dostupan

Kada u tabeli menjamo slog, automatski se menjaju i podaci ispod. Pošto su sve kontrole vezane za isti izvor podataka (Eng: DataSource), u našem slučaju na prvu tabelu Ds objekta, logično je da je prikaz sinhronizovan. Takođe ćete primetiti da možete menjati podatke direktno ili u tabeli ili u kontrolama ispod nje. Te izmene se takođe automatski prikazuju. Međutim kada zaustavite, pa ponovo pokrenete primer primetićete da su podaci ostali neizmenjeni. Sve izmene koje se na ovaj način rade su lokalnog tipa, odnose se samo na DataSet objekat koji je u memoriji računara. Baza podataka se ne ažurira automatski. Da bi ovo uradili neophodno je formirati Insert, Update i Delete komande što u ovom momentu prevazilazi ambicije ovog početničkog serijala.

Za kraj ovog nastavka i serijala, pokušajte sledeće:

Sama forma takođe poseduje DataBindings kolekciju. Ako želimo da u naslovu forme prikažemo naziv proizvoda, dovoljno je setiti se da referencu na formu predstavlja ključna reč this.

Dopišite još jednu liniju koda:

```
this.DataBindings.Add("Text", Ds.Tables[0],  
"ProductName");
```

Sada pokrenite primer i videćete kako se i naslov forme menja u zavisnosti od aktivnog zapisa.

MM