Zoran Ćirović
Ivan Dunđerski

TEHNIKE VIZUELNOG PROGRAMIRANJA

C#

Viša elektrotehnička škola
Beograd 2005.

Recenzenti
Slobodanka Đenić

Izdavač
Viša elektrotehnička škola Beograd

Za izdavača
Dragoljub Martinović

Dizajn
Snežana Trstenjak

Lektor i korektor
Anđelka Kovačević

CIP – Katalogizacija u publikaciji
Narodna biblioteka Srbije, Beograd

004.42’236:004.738.1(075.8)
004.432.2C#(075.8)

ĆIROVIĆ, Zoran
Tehnike vizuelnog programiranja : C# / Zoran Ćirović, Ivan Dunđerski. – Beograd: Viša elektrotehnička škola, 2005 (MST Gajić). – 241 str.: ilustr.; 24 cm

Tiraž 250. – Bibliografija: str.187.

ISBN 86-85081-29-7
1. Dunđerski, Ivan
a) Vizuelno programiranje - .NET tehnologija b) Programski jezici “C#” - .NET tehnologija
COBISS.SR-ID 120683276

Predgovor

Ova knjiga namenjena je studentima Više elektrotehničke škole u Beogradu, kao priručnik za predmet „Tehnike vizuelnog programiranja“.

Pokazane su tehnike i pravci u razvoju novih aplikacija kroz veliki broj primera. Svi primeri rađeni su na razvojnom okruženju - *Microsoft VisualStudio.NET*.

Knjiga može poslužiti svima koji žele da nauče osnovne tehnike rada na .NET platformi, odnosno C# programskog jezika. Gradivo je prilagođeno čitaocima koji vladaju osnovama OOP-a, pa je naglasak na vizuelnom programiranju.

U Beogradu, januara 2005. godine Autori

SADRŽAJ

1. OSNOVE .NET PLATFORME 1

Poređenje: C# vs. C++/Java 2
C# i Java 3
C# u odnosu na C++ 4
"Hello World" 5
Proverite da li je .NET framework instaliran 5
Komentari 6
Mala/velika slova 6
Metod *Main* 7
Prevođenje (kompajliranje) koda 7

2. UVOD U VISUAL C#.NET 9

IDE – Integrisano Razvojno Okruženje 9
Toolbar (paleta sa alatima) 11
Toolbox (kutija sa alatima) 12
Solution Explorer, Class View, Resource View, Properties 13
Class View 15
Dodavanje nove klase projektu 16
Dodavanje metoda i polja 18
Tipovi aplikacija 20
Promenljive u C# 20
Vrednosni tip promenljive 21
Referentni tip promenljive 22
Svojstva u C# 23
Dodavanje svojstava iz IDE 25

3. DIZAJN KONTROLA 29

Osobine Windows kontrola 29
Relacija *roditelj-dete* 30
Koordinatni sistem 30
Postavljanje kontrole na formu 31
Selektovanje kontrole, podešavanje pozicije i veličine 32
Zajednička svojstva Windows kontrola 33
Name 33
Location 34
Size 34
Bounds 35

Text 35
Visible 36
Enabled 36
Fokus kontrole i aktivan prozor 36
TabIndex i *TabStop* 37
Programsko kreiranje kontrola 38
Postavljanje kontrole u fokus 42

4. DOGAĐAJI 43

Spy++ 43
Delegati 45
Događaji i IDE 47
Događaji vezani za forme i kontrole 51
Paint 52
Resize 52
Događaji tastature 52
Događaji miša 53

5. FORMA 55

Izgled forme 55
Naslovna linija - *Title bar* 56
Ikona - *Icon* 56
Sistemska dugmad 59
Klijentska oblast 61
BackgroundImage 62
WindowState 62
ShowInTaskbar 62
Kreiranje objekta forme 62
Zatvaranje forme 63
Pokretanje forme 63

6. WINDOWS KONTROLE 67

Dugme 67
Text 67
TextAlign 68
DialogResult 68
Image, ImageAlign, ImageList, ImageIndex 68
FlatStyle 69
Radio-dugme 69
Checked 70
CheckAlign 70

Appearance 70
Polje za potvrdu 71
Checked 72
Alignment 72
CheckState 73
ThreeState 73
Appearance 74
TEKST kontrole 74
Labela 74
TextBox 75
Kombo polje 75

7. APLIKACIJE SA VIŠE FORMI 79

Aplikacije sa višestrukim dokumentima - MDI 85

8. KONTEJNERI KONTROLA 89

Forma 89
Panel 89
Osobine panel kontrole 89
GroupBox 90
Stranice svojstava 90
Kontrola Property Sheet 93
Dock 95
ImageList 96
MultiLine 97
Alignment 99
Appearance 99
Text 100

9. DIJALOZI 101

Dialog Box 101
Kreiranje 102
AcceptButton 102
CancelButton 103
Vrste dijaloga 103
Modalni dijalozi 103
Nemodalni dijalozi 104

10. MENIJI 107

Glavni meni 107
Kreiranje 107
Komande menija sa tastature 108
Prečice 108
Dodavanje akcija 111
Meniji MDI aplikacija 112
Kontekstni meni 113

11. *TOOLBAR* 117

Osobine i kreiranje 117
Programiranje 121

12. UVOD U GRAFIKU 123

Klasa *Graphics* 124
Strukture 127
Klasa *Pen* 128
Klasa *Brush* 131

13. UVOD U XML 133

XML deklaracija 133
XML Tag 134
Elementi XML-a 135
XML atributi 136
Visual C# i XML 137
DataSet 137
Kreiranje XML fajla 138
Kreiranje XML datoteke editorom teksta 138
Kreiranje XML fajla pomoću VS.Net 139
Kreiranje XML koda pomoću klase *XmlDocument* 141
XML Atributi 143

14. WEB-SERVISI 149

Komunikacija sa aplikacijom 149
Traženje Web servisa 150

Metode Web servisa 151
.NET okruženje i Web servisi 151
Kreiranje Web-servisa 151
Dodavanje Web-metoda 153
Testiranje 154
Implementacija Windows klijenta 155

15. OSNOVE ADO.NET-A 160

ODBC 160
Data Source Name 160
Konekcija 162
DataAdapter 166
DataSet 170
DataGrid 173
Pristupanje podacima objekta *DataSet* 173
Promena podataka u bazi 174
Objekti *Command* i *DataReader* 175

16. .NET SKLOPOVI 177

Osobine 177
Samoopisivanje 177
Instalacija sklopa 177
Komponente 177
Sastav sklopova 178
Implementacija sklopa pomoću IDE 178
MSIL Disassembler 180
Atributi sklopova 183
Verzija sklopa 185
Kompatibilnost verzija 185
Privatni i deljeni sklopovi 186

17. LITERATURA 187

PRIMERI 189

1. OSNOVE .NET PLATFORME

.NET (dot-net) je ime najmodernije platforme koje Microsoft vezuje za softverske tehnologije budućnosti. U nazivu je osnovna poruka koju nosi ova tehnologija - **dostupnost u svakom trenutku na svakom mestu**. Ova platforma predstavlja dugoročni i strategijski plan razvoja ne samo u Microsoftu. .NET daje realne osnove da postane osnovna platforma razvoja modernih aplikacija. Radni okvir .NET razvijen je sa ciljem da obezbedi okruženje za razvoj svih modernih aplikacija na Windows operativnom sistemu.

Jedna od osnovnih osobina ove platforme je njena **orijentacija ka distribuiranim aplikcijama preko Interneta**. Ovo sa sobom nosi obavezno još prednosti:

- Distribuirane aplikacije su **više objektno orijentisane**;
- Ovaj stil programiranja ubrzava stvaranje **kolekcije specifičnog koda na jednom mestu, nasuprot dosadašnjem stilu gde se stvaraju redundantne kopije na mnogo mesta**. (Redundantnost je pojam koji se koristi u tehnici kada je reč o nepotrebnom višku ili preobimnosti.) Ovo svakako povećava efikasnost i opredeljuje za ovakve aplikacije;
- Ovaj tip apikacije pruža softverske celine raspoložive različitim uređajima preko interfejsa;
- Kontrolisanjem pristupa u realnom vremenu (*real-time*) ka distribuiranim čvorovima (delovi jedne softverske celine) moguće je lakše kontrolisanje rada takvih aplikacija. **Ovaj pristup pomera aplikacije od objektno orijentisanih u pravcu '*services provided*';**
- Biblioteka klasa razvijena je od samog početka koristeći dragocena iskustva. Ovo daje veoma lepu osobinu, a to je dobar dizajn, dosledno i dobro definisanje osnovnih tipova;
- Jezička nezavisnost. Ovo je postignuto postojanjem međujezika, IL (Intermediate Language, ili MSIL), tj. kôd napisan na bilo

kom jeziku koji ima podršku za .NET prevodi se na kôd razumljiv tom međujeziku;

- Podrška za Web (XML) servise. .NET ima razvijene alate za jednostavno pisanje XML servisa;
- Poboljšani pristup dinamičkim Web stranicama baziran na ASP.NET tehnologiji;
- Efikasniji pristup podacima preko ADO.NET klasa;
- .NET postavlja i novi pristup za zajedničko korišćenje koda. Nasuprot tradicionalnih *dll* biblioteka, uvodi se koncept **sklopova** (*assembly*);

Zajedničko funkcionisanje različitih platformi imperativ je u distribuiranim aplikacijama. Zato je bilo neophodno **obezbediti standardizaciju u razmeni podataka**. Tako je .NET platforma sagrađena na standardima tehnologije **XML** i SOAP. Ukoliko se prvi put srećete sa pojmom XML ne brinite. Rad sa XML je veoma dobro podržan u .NET-u i prilično olakšan upotrebom radnog okruženja.

Rad takvih aplikacija podrazumeva da postoji **.NET Framework**. *.NET Framework* sadrži **CLR** (*Common Language Runtime*) kao i kolekcije klasa ovog okruženja. Posebno ističemo ASP.NET (nova generacija *Active Server Pages* tehnologije) i WinForms (služi razvoju desktop aplikacija).

Još par detalja o CLR...

CLR izvršava kôd koji je kompajliran na .NET platformi. CLR specifikacija je otvorena za sve platforme, uključujući i non-Windows. Takođe, svi programski jezici mogu se koristiti za rad sa .NET klasama. Ove karakteristike obezbeđuju pravljenje distribuiranih aplikacija čije celine mogu da rade na nezavisnim platformama i čak pisane različitim programskim jezicima.

Svi jezici koji su obuhvaćeni novim paketom *Microsoft VisualStudio.NET* imaju podršku za .NET klase. Kako svi oni imaju "ispod" CLR to ih čini ravnopravnim više nego ikada. Međutim, C# je posebno napravljen novi programski jezik upravo za .NET. Drugi programski jezici na ovoj platformi su postojali i pre nastanka .NET. To ovom programskom jeziku omogućava optimalno korišćenje .NET okruženja.

Poređenje: C# vs. C++/Java

Za nekog ko tek treba da počne sa učenjem novog programskog jezika uvek je aktuelno pitanje zbog čega treba da uloži napor za njegovo savladavanje i šta tim ulaganjem dobija. Danas su, više nego ikada, programski jezici moćni, tako da kvalitet softvera zavisi najviše od vrednosti programera. Ipak, razlike i sličnosti između jezika postoje i dobro ih je poznavati. Zbog aktuelnosti izvešćemo poređenje jezika C# sa Javom i C++.

C# i Java

- C# i Java svrstavaju se u naslednike jezika C. Pripadaju jezicima nove generacije i uključuju savremene osobine. Na primer, oba jezika imaju "*garbage collection*" (skupljač memorijskih otpadaka u upravljanoj dinamičkoj memoriji) koji oslobađa programera razmišljanja o delovima programiranja niskog nivoa. Sintaksno jezici su slični.

- Osnovna zajednička osobina za oba jezika je da se prevođenje (kompajliranje) vrši do međujezika *Intermediate Language* (*IL*). Za C# taj jezik se naziva *Microsoft Intermediate Language (MSIL)*, dok je za Javu to *Java bytecode*. Za oba jezika IL može biti startovan kao interpretator ili *just-in-time* (*JIT*) prevodilac. C# ipak obezbeđuje dalje prevođenje u prirodan - mašinski kod *native code*. Za razliku od *Java bajtkoda* koji se najčešće interpretira, u .NET IL uvek se prevodi u pravo vreme. Na ovaj način čuvaju se performanse aplikacije. Takođe, ne prevodi se cela aplikacija odjednom već samo oni delovi koji se pozivaju u toku izvršavanja programa (run-time).

- Dalje, C# sadrži više osnovnih tipova podataka od Jave.

- Takođe C# sadrži enumeratore i preklapanje operatora za korisničke tipove.

- Kao i Java, C# napušta višestruko nasleđivanje koje je postojalo u C++. Osobina polimorfizma je nešto usložnjena dodavanjem opcija *overriding/hiding* (kasnije će biti detaljnije objašnjene).

- C# uvodi novu vrstu objekata koja ima puno sličnosti sa klasama tzv. delegate (*delegates*). Delegati sadrže detalje nekog metoda (slično kao pokazivači na funkcije u C++ jeziku).

- Za razliku od Jave koja višedimenzionalne nizove implementira preko jednodimenzionalnih (drugim rečima pravi nizove nizova), C# sadrži prave pravougaone nizove.

C# u odnosu na C++

Mada su sličnosti sa VB i Javom očigledne, ipak C# je najbliži C++.

- Jedna od najvažnijih promena u odnosu na C++ je u tome da se zahteva fajl sa zaglavljima. Novi pristup je dobijen korišćenjem tzv. *folding editora*, koji može da "sklopi i rasklopi" kôd metoda, odnosno da ga prikaže kao deklaraciju ili kao definiciju metoda. (Ovaj editor može to isto da uradi i sa komentarima.)

- ".NET runtime" na kome se izvršava C# sadrži upravljanje memorijom upotrebom objekata "*garbage collector*". Zbog toga je korišćenje klasičnih pointera u C# manje važno nego u C++. Klasični pointeri mogu se koristiti u C#, ali samo gde je kôd označen kao 'unsafe' i to pre svega ima smisla kada su performanse najvažnije.

Postoje još neke izmene u odnosu na C++. Na primer:

-Odsustvo globalnog prostora, sve je u klasama;

-Sve je izvedeno iz praklase Object. Ovo važi i za vrednosne i za referentne tipove. To je preduslov da funkcioniše garbage collector;

-Nema višestrukog nasleđivanja klasa, samo interfejsa;

-Sigurnost tipova: garantuje se da će u svakoj promenljivoj biti vrednost tipa za koji je promenljiva deklarisana;

-Pravilo izričite dodele promenljivoj pre korišćenja, u skladu sa sigurnošću tipova (obavezna inicijalizacija).

Suštinska razlika je prisustvo CLR tj. IL za C#, o čemu je već bilo govora.

ANSI C++ se bazira na tehnologiji koja je stara više od 10 godina, tako da nema podršku za *unicode* (pogledajte www.unicode.org) ili za generisanje xml dokumentacije.

Mada C++, Java i C# spadaju među 10 najpopularnijih programskih jezika, pravo poređenje se svodi na aktuelnost pojedinih programskih jezika. U poslednjih par godina jedino je C# uspeo da zabeleži porast popularnosti. Pre svega ovo je izbor onih koji vole C jezik, ali i onih koji su naučili da programiraju na C/C++, a želeli bi da nauče još više o novim tehnologijama. Budućnost pripada jeziku C#.

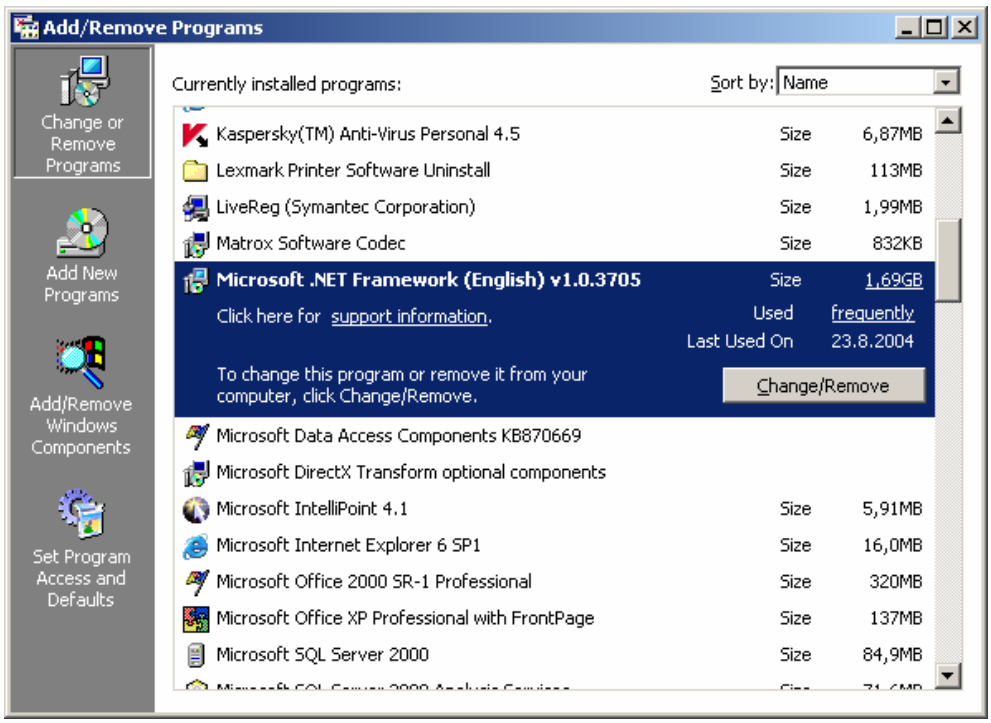
"Hello World"

Po dobrom starom nepisanom pravilu, svi primeri počinju tradicionalnim programom *Hello World*. U ovom poglavlju videćemo šta nam je sve potrebno da bismo krenuli sa programiranjem u C#.

Proverite da li je .NET framework instaliran

Pre svega da bismo mogli da koristimo C# ili .NET klase na nekoj mašini moramo imati instaliran .NET framework SDK. Ukoliko ste instalirali Visual Studio .NET ne brinite, instalacijom ovog paketa vaša mašina sadrži i .NET framework.

Ukoliko niste sigurni šta vaša mašina sadrži, pogledajte u *Control Panel* – u spisak instaliranih programa. Na slici ispod vidi se primer postojanja jedne verzije platforme *.NET Framework*.



Da biste nešto i napisali potreban vam je jedan najobičniji editor teksta. Kôd je dat ispod.

```
using System;
public class HelloWorld
{
   public static void Main()
   {
      // komentar u jednoj liniji
      /*
         komentar
         vise
         linija
      */
      Console.WriteLine( "Hello World! );
   }
}
```

Iz primera treba uočiti sledeće karakteristike C# jezika.

Komentari

Iz koda se vidi i način na koji mogu da se pišu komentari u C#. Sintaksa je uobičajena i za C++. Komentari se dele na linijske i oni na početku imaju dve kose crte. Druga vrsta komentara se proteže između znakova /* odnosno */ i prenosi se na više linija.

Mala/velika slova

C# pravi razliku između malih i velikih slova - ***case-sensitive***. Ako bi u prethodnom primeru napisali "console" (malim početnim slovom) umesto "Console" (velikim početnim slovom) program se ne bi preveo, a prevodilac bi prijavio grešku.

Napomena: Odvajanje naredbi u kodu je identično kao u programskom jeziku C.

C# je objektno orijentisan programski jezik i **sve naredbe moraju stajati u klasama**. Drugim rečima, zapamtite da C# nema globalnog

prostora, kao što je to u C++ jeziku. Klasa u kojoj je smešten ceo kôd ovog primera zove se "HelloWorld" i ima jednu jedinu naredbu:

```
Console.WriteLine( "Hello World!" );
```

Osim imena klase , na samom vrhu, treba primetiti liniju

```
using System;
```

Ova naredba omogućava da se nadalje u kodu koriste sve metode iz prostora imena *System,* a da eksplicitno ne navedete reč System. Tako se klasa *Console* sa metodama za rad sa konzolom, koja pripada prostoru imena *System*, može pozvati

```
Console.WriteLine("….")
```

Umesto

```
System.Console.WriteLine("….")
```

tzv. puna kvalifikacija imena.

Metod *Main*

U klasi HelloWorld nalazi se metod Main. Obratite pažnju da je prvo slovo veliko čime se pravi razlika u odnosu na C funkciju main. Ova funkcija je deklarisana kao **static** i **public**. Program može imati samo jednu ulaznu tačku (*entry point*) i ona se ostvaruje upravo preko ovog metoda.

Prevođenje (kompajliranje) koda

Da biste preveli ovaj kôd potrebno je da pozovete kompajler sa komandne linije ili koristite integrisano okruženje. Kao primer pokazaćemo upotrebu linijskog kompajlera za ovaj kôd.

U jedan DOS prozor unesite:

```
csc HelloWorld.cs
```

Ako se kompajliranje uspešno odradilo u istom direktorijumu naći ćete fajl

helloworld.exe

Pokretanje ovog programa vrši se ako otkucate samo ime (sa ili bez ekstenzije) a zatim pritisnete ENTER.

Kompajliranje iz okruženja je jednostavno i radićemo ga u narednim primerima.

Primer: Postavljanje kompajlera u "*path*"

Verovatno da ste imali problem sa pokretanjem kompajlera. Ukoliko imate instaliran .NET Framework morate postaviti folder u kome se nalazi kompajler ***csc*** u putanju koja će se pretraživati pri startovanju nekog programa. Ove putanje označene su posebnom promenljivom okruženja koja se naziva PATH.

1. Kliknite na *Start*->*Control Panel* oo
2. Zatim kliknite na *System*
3. Na dijalogu *System Properties* kliknite na tab *Advanced,* a zatim na *Environment Variables*
4. U sekciji *System Variables* odaberite *Path* a zatim kliknite na *Edit*
5. Dodajte na kraju ; sa putanjom do direktorijuma gde se nalazi csc.exe. Na primer:

   %SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;C:\WINDOWS\Microsoft.NET\Framework\v2.0.40607

6. Potvrdite pritiskom na OK

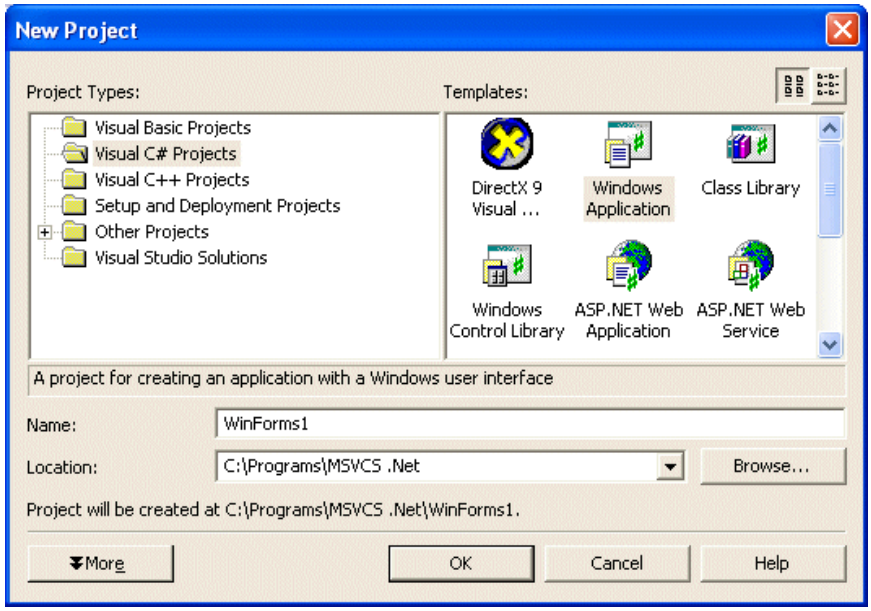
2. UVOD U VISUAL C#.NET

IDE – Integrisano Razvojno Okruženje

Visual Studio .Net IDE (*Integrated Development Environment*) predstavlja izuzetan interfejs za stvaranje i testiranje aplikacija. Tipovi aplikacija koje VS.NET podržava su najrazličitiji i ne postoji jedinstven šablon za sve. Zbog toga se u projektovanju nove aplikacije ona mora pridružiti nekoj od grupa.

Primer: Kreiranje novog projekta korišćenjem osnovnog menija.

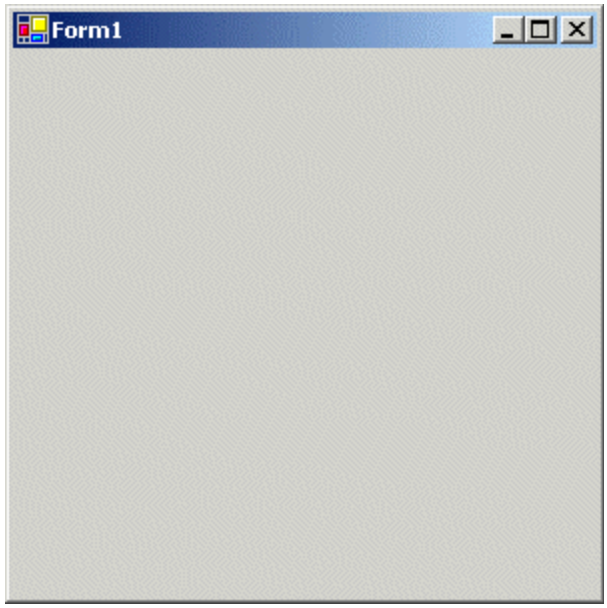
1. Izaberite ***File->New->Project...***
2. Na ***Project dialog*** prozoru izaberite ***Visual C# Projects*** u listi ***Project Types***
3. Na listi ***Templates*** izaberite ***Windows Application***
4. U tekst polje ***Name*** unesite ime vašeg projekta ili možete prihvatiti već ponuđeno ime, na primer *WinForms1*



5. Potvrdite unos pritiskom na dugme ***OK***. Nova aplikacija će biti kreirana sa podrazumevanom formom.

6. Da biste preveli i pokrenuli vašu aplikaciju odaberite iz glavnog menija

   Debug->Start Without Debugging



7. Test aplikaciju možete zatvoriti pritiskom na dugme  i vraćate se u programsko okruženje.

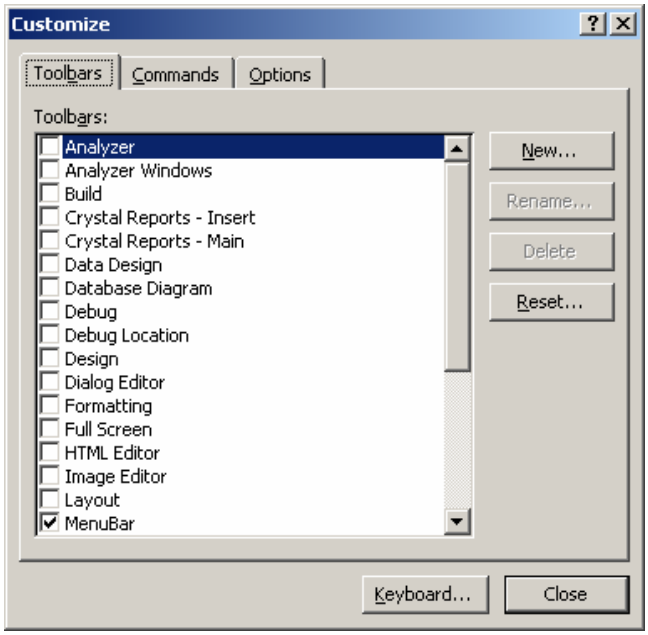
Toolbar (paleta sa alatima)

Inače, ispod glavnog menija nalazi se paleta sa dugmićima (na kojima su slike koje asociraju na funkciju) koji pokreću neke najčešće korišćene komande. Ta paleta se naziva ***toolbar***.

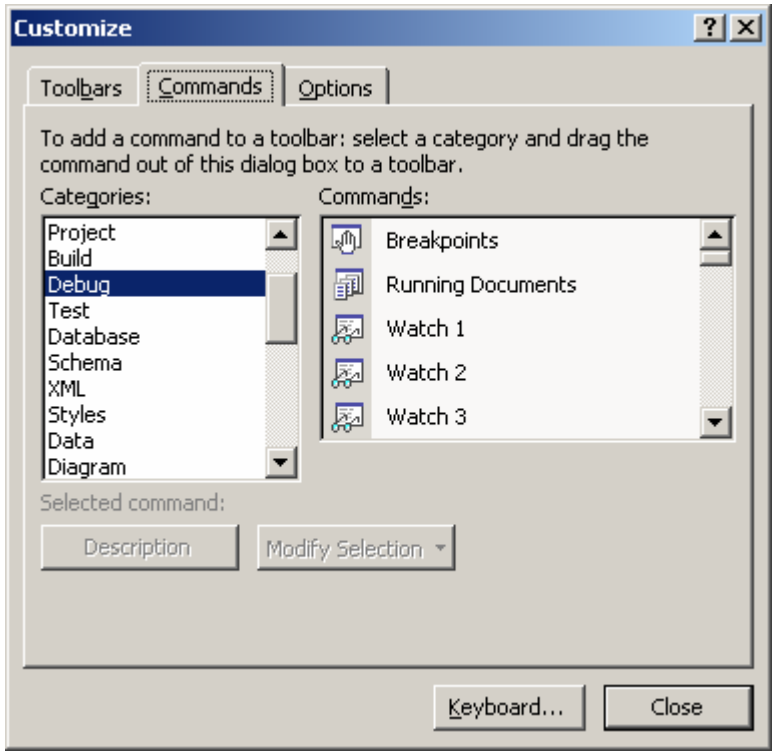
Postoji više tipova kontrole *toolbar*. Ako pored njega kliknete desnim dugmetom miša videćete spisak sa ozačenim objektima toolbar koji su aktivni. Sličice koje vidite pripadaju tzv. standardnom *toolbar* objektu.

Bar nazivamo posebnim prozorom, koji je deo aplikacije i na kome se uobičajeno nalazi neka paleta alatki. Takođe, uobičajeno je da se ta paleta "lepi" za neku stranu prozora glavne aplikacije, ali može stajati i samostalno.

Ako sa glavnog menija odaberete ***Tools->Customize*** možete podešavati alatke koje će biti na radnoj površini ili dodavati nove. Ovo podešavanje izvodite na tabu (kartici) ***Toolbars***, kao što je prikazano na slici ispod.



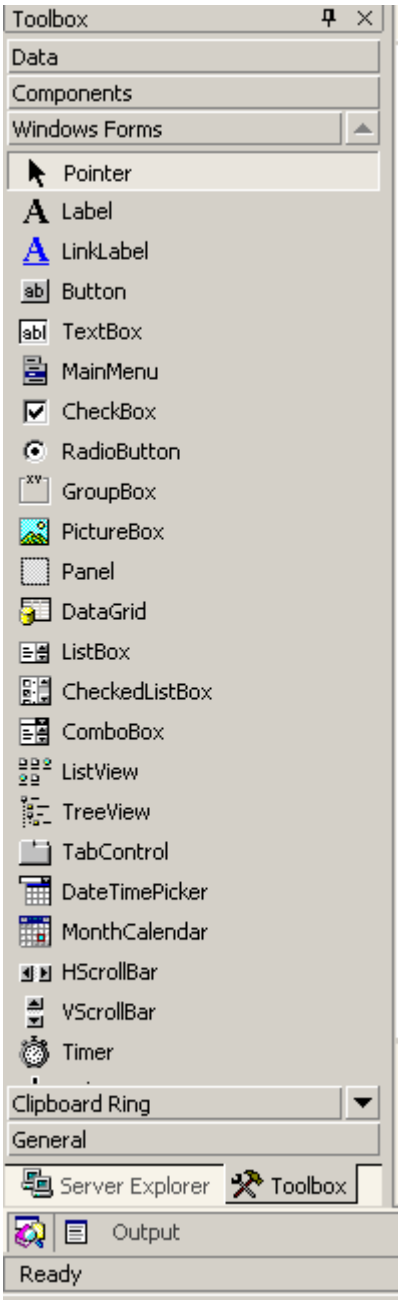
Na kartici ***Commands*** možete dodavati sopstvene komande nekom od objekata *toolbar*.



Kao što je i pomenuto radno okruženje se može podesiti (customize) po slobodnoj volji, tako da pruža mogućnost programeru da se u njega lako uklopi i "udobno smesti".

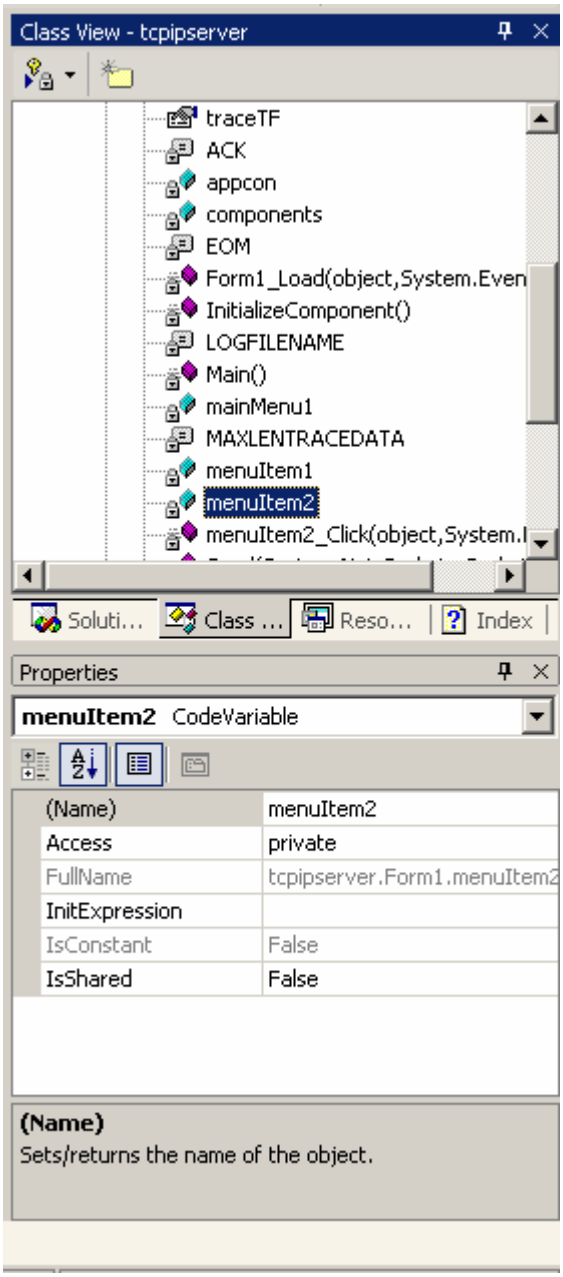
Toolbox (kutija sa alatima)

Uz levu ivicu ekrana nalazi se bar (traka) sa natpisom ***Toolbox*** (kutija sa alatima dizajnera) vertikalno orijentisan. U objektu *toolbox* razvrstane su kontrole po kategorijama. Na liniji gde je naslov postoji dugme pribadača (sa skraćenim opisom *Auto hide*) , kojim se menja ponašanje ove kontrole. Pritiskom na pribadaču bar može biti fiksiran za ivicu glavnog prozora. Ukoliko je pribadača u gornjem položaju bar je podešen tako da nestaje kada se miš pomeri sa njega. Ovo je podrazumevano ponašanje okruženja, sa idejom da se poveća korisna površina ekrana kada se kutija sa alatima ne koristi.



Solution Explorer, Class View, Resource View, Properties

Na desnoj strani ekran je podeljen u dve sekcije. Gornja sekcija uključuje različite tabove, kao što su *Class View*, *Resource View* i *Solution Explorer*. U donjoj sekciji na desnoj strani nalazi se prozor koji prikazuje svojstva stavki koje su selektovane na gornjoj.

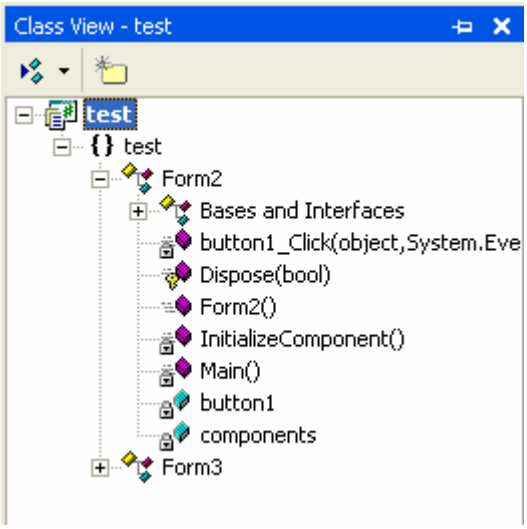


Naravno, još jednom da ponovimo, izgled radne površine je potpuno proizvoljan i moguće ga je prilagođavati sopstvenim željama. Opis koji je dat važi za inicijalni izgled IDE okruženja.

Zbog značaja koji ima u kreiranju aplikacija sada ćemo pokazati neke mogućnosti koje nam pruža *Class View*. Upotrebu ostalih alata pikazaćemo kasnije na konkretnim primerima.

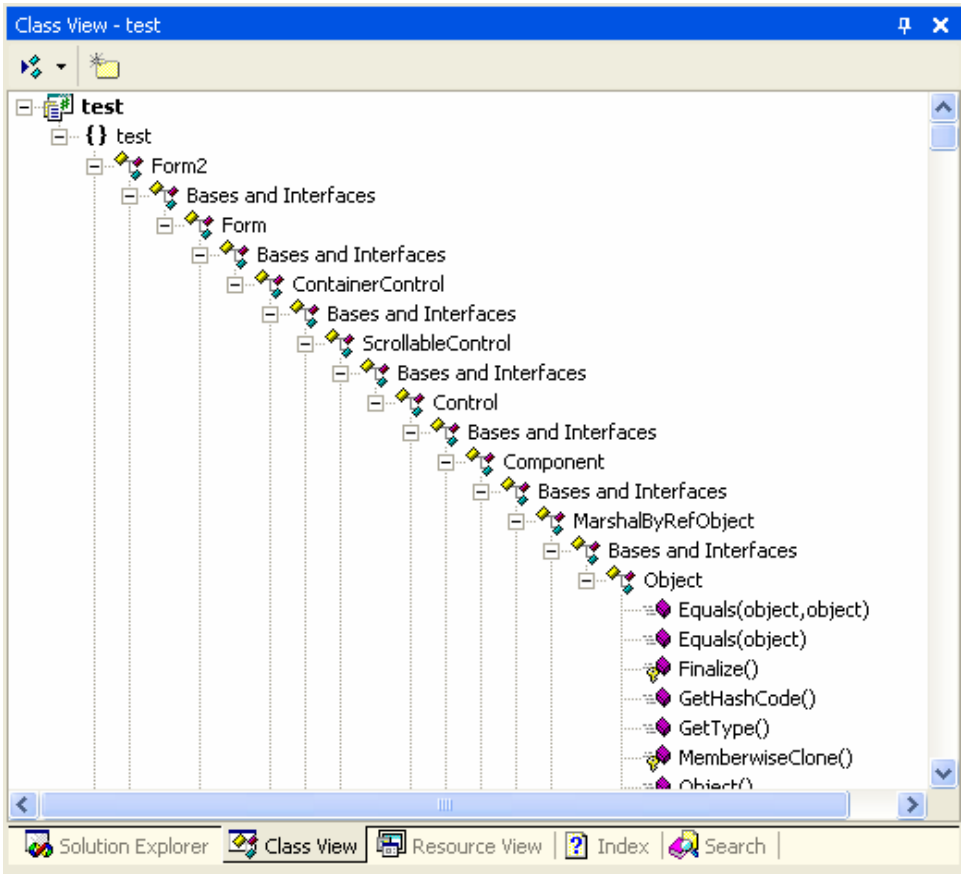
Class View

Class View daje prikaz klasa u projektu koji stvarate i ujedno vam omogućava da dodajete i menjate postojeće klase. Prikaz klasa dat je u obliku stabla, dakle hijerarhijski uređenih. Na slici je dat jedan primer projekta *test*.



Čvor najvećeg nivoa predstavlja sam projekat *test*. Sledeći nivo čvorova su prostori imena iz projekta. U gornjem primeru postoji samo jedan. U prostoru imena, koji se takođe naziva *test,* sadržane su dve klase: Form2 i Form3. Klasa Form2 sadrži dva privatna polja *button1* odnosno *components*, zatim privatne metode *Main()*, *InitializeComponent()* i *button1_Click (...).* Klasa *Form2* sadrži i konstruktor *Form2*(), zaštićenu metodu *Dispose*(*bool*).

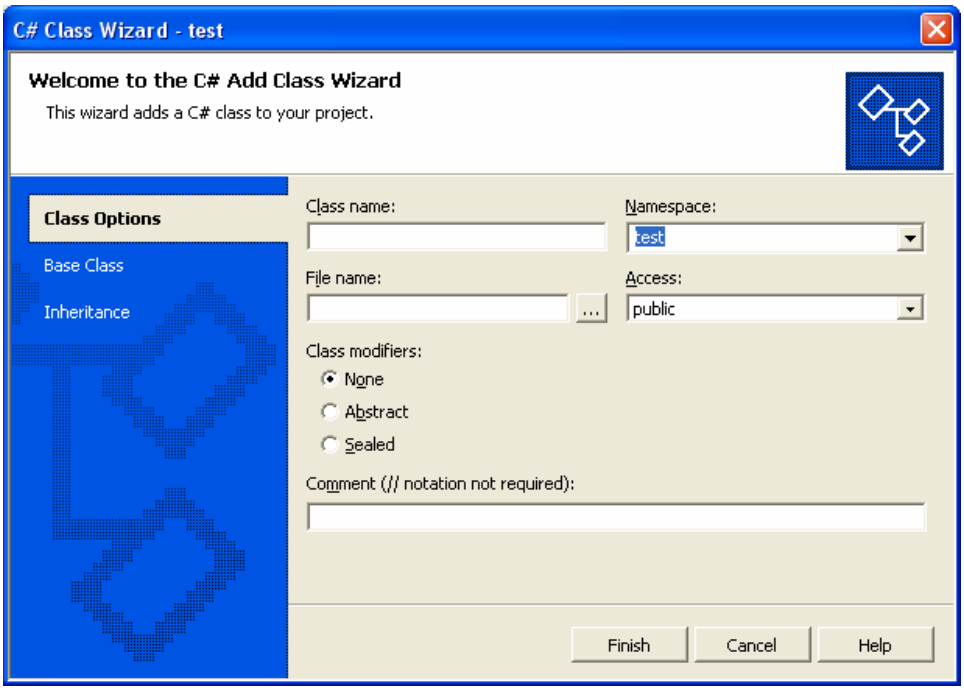
Ako proširimo čvorove *Bases and Interfaces* videćemo da je u korenu klasa *Object.* Ova klasa se nalazi u korenu svih klasa jezika C#. Ovo smo pominjali kada smo nabrajali prednosti C# jezika.



Dodavanje nove klase projektu

Pomoću Class View prikaza možete dodati novu klasu vašem projektu. U ovom prikazu pritisnite desnim tasterom miša projekat i odaberite stavku u meniju ***Add Class.*** Drugi način je izborom stavke menija *Project->Add Class.*

Pojavljuje se čarobnjak za kreiranje klasa – *C# Class Wizard*, kao na slici ispod



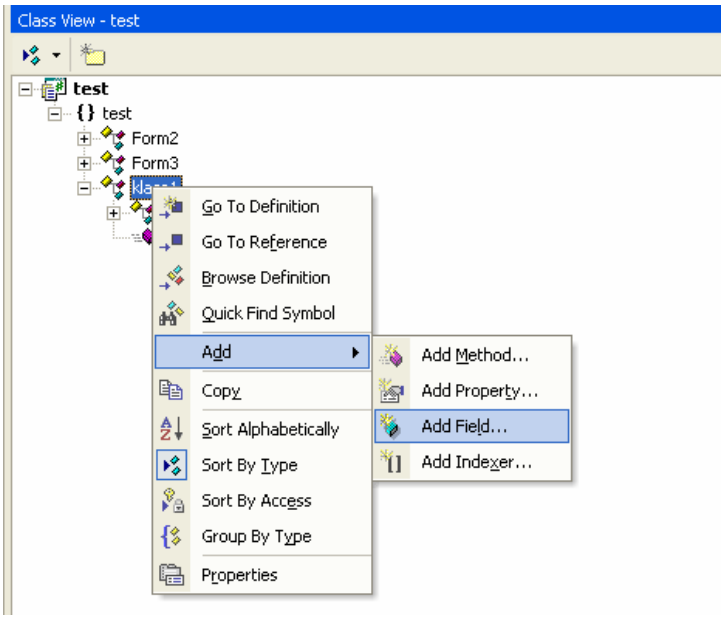
Značenje polja dato je u tabeli ispod.

Naziv polja	Opis
Class Name	Ime nove klase
Namespace	Naziv prostora imena kome se klasa pridružuje
File Name	Ime fajla u kome će biti smeštena definicija klase
Access	Definiše mogućnost pravljenja referenci u drugim delovima i drugim aplikacijama
Class Modifiers	Apstraktne i zapečaćene klase kontrolišu korišćenje klase u vezama nasleđivanja.

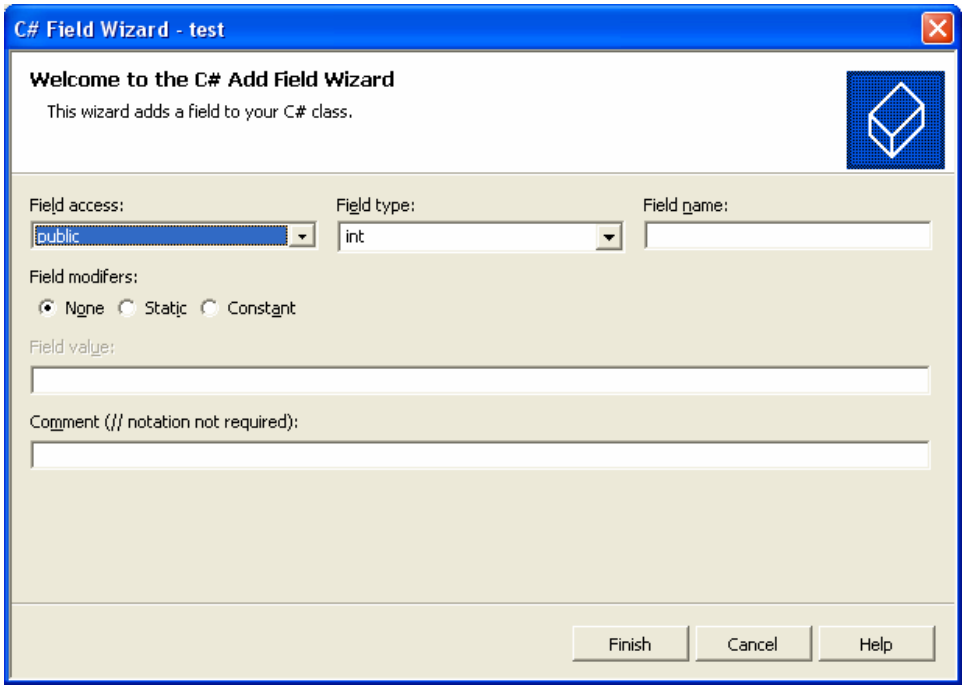
	Kasnije će biti govora o ovome.
Comment	Komentar koji se kasnije i vidi u IDE editoru preko *Intelli-Sense* programa.
Kartica ***Base Class***	Definisanje osnovne klase za klasu koju kreirate. Dostupne su ne samo klase iz .NET Frameworka već i klase iz vašeg projekta.
Kartica Inheritance	Definisanje interfejsa koje implementirate u vašoj klasi.

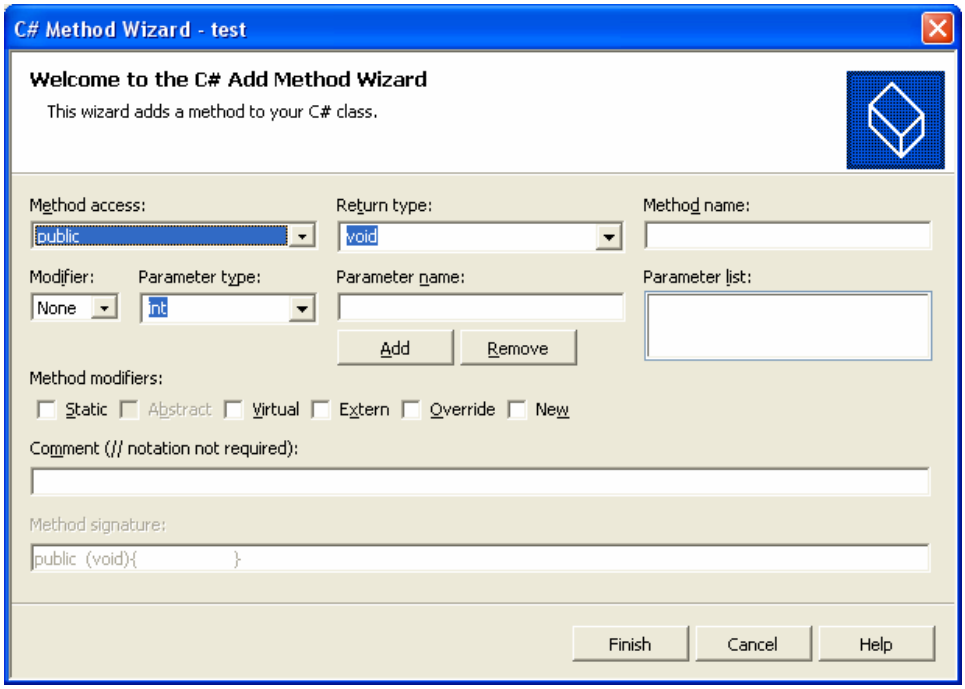
Dodavanje metoda i polja

Na sličan način kao što smo dodali novu klasu projektu možemo dodati novo polje jednoj klasi. Dakle, koristeći *Class View* pritiskom na desni taster na klasu kojoj želite da dodate novo polje dobijate meni gde birate stavku *Add* a zatim birate stavku koju dodajete klasi, kao na slici ispod.



Na slikama ispod dati su čarobnjaci za dodavanje novog polja klasi, odnosno za dodavanje novog metoda.





Ne zaboravite da ovi čarobnjaci služe da programeru olakšaju posao. Sve što oni urade možete uraditi i ručno ukoliko niste zadovoljni kodom koji generišu ili smatrate (volite) da ovo treba da radite ručno.

Tipovi aplikacija

Pre nego što nastavimo sa novim stvarima vratimo se na primer kreiranja novog projekta. Na listi *Templates* odabrali smo *Windows application*. Ovo smo uradili prećutno, sa namerom da što pre kreirate vašu .NET Windows aplikaciju i osetite bar na trenutak mogućnosti okruženja koje je ispred vas. Sada ćemo pojasniti neke od opcija koje vam stoje na raspolaganju.

Šablon (Template)	**Kreira se...**
Windows Application	Windows aplikacija sa jednom praznom početnom formom, sa svim osobinama koje takva aplikacija mora da ima.
Console Application	aplikacija koja se izvršava iz komandne linije ili sa konzolnog prozora.
Empty Project	ništa. Sav kod morate sami uneti u aplikaciju.
Windows Service	servis koji se izvršava u pozadini.
Class Library	.NET klasa koja se može pozvati iz drugog koda.
ASP.NET Web Service	klasa koja radi kao Web servis
ASP.NET Web App.	stranice i klase koje generišu HTML odgovor poslat pretraživaču sa ovih stranica.

Promenljive u C#

Postoje dve vrste promenljivih u C#. Promenljive mogu biti **vrednosnog tipa, i referentnog tipa**.

Vrednosni tip promenljive

Ove promenljive sadrže podatak. One su slične kao *primitive types* u Javi. Svi tipovi su izvedeni iz klase Object. Vrednosne promenljive ne mogu da sadrže null vrednost za razliku od referentnih tipova. Ove promenljive se čuvaju na steku.

C# Type	.NET Framework Type	Označen	Dužina [bajt]	Opseg vrednosti	*Default*
sbyte	System.Sbyte	Da	1	-128 do 127	0
short	System.Int16	Da	2	-32768 do 32767	0
int	System.Int32	Da	4	-2147483648 do 2147483647	0
long	System.Int64	Da	8	-9223372036854775808 do 9223372036854775807	0L
byte	System.Byte	Ne	1	0 do 255	0
ushort	System.UInt16	Ne	2	0 do 65535	0
uint	System.UInt32	Ne	4	0 do 4294967295	0
ulong	System.UInt64	Ne	8	0 do 18446744073709551615	0
float	System.Single	Da	4	7 cifara preciznost	0.0F
double	System.Double	Da	8	15-16 cifara preciznost	0.0D
decimal	System.Decimal	Da	12	28-29 cifara preciznost	0.0M
char	System.Char	N/A	2	6-bitni *Unicode* karakter	'\0'
bool	System.Boolean	N/A	1	*true* ili *false*	*false*

Primer upotrebe vrednosnih promenljivih.

```
int x = 99;
int y = x;
x = 138;
```

U gornjem primeru prikazana je osnovna karakteristika vrednosnih tipova. Promenljiva y čuva kopiju vrednosti koja je inicijalno postavljena u promenljivoj x, tj. sadrži vrednost 99. Ako se vrednost promenljive x promeni na 138 u sledećoj liniji koda ovo nema uticaja na vrednost promenljive y.

Referentni tip promenljive

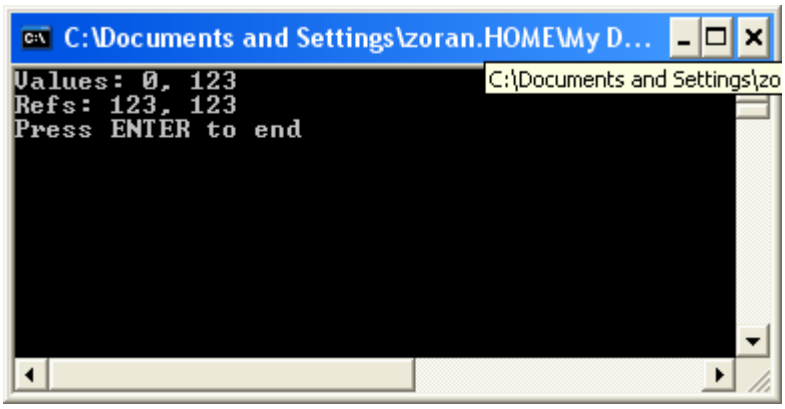
Ovaj tip promenljivih čuva reference na memorijske adrese sa vrednostima i one se alociraju na tzv. *heap* memorijskom delu. Heap je deo memorije u kome se dinamički alociraju promenljive. Ovaj tip promenljivih poznat je još i kao objekti. Ovo je razlika u odnosu na C++. Ove promenljive čuvaju reference na stvarne podatke tj. na stvarni objekat u memoriji.

Primer: Kôd koji prikazuje razlike vrednosnog i referencijalnog tipa promenljive.

```
using System;

namespace Project1
{
  class Class1
  {
    public int Value = 0;
  }
  class Test
  {
    static void Main()
    {
      int val1 = 0;
      int val2 = val1;
      val2 = 123;
      Class1 ref1 = new Class1();
      Class1 ref2 = ref1;
      ref2.Value = 123;
      Console.WriteLine("Values: {0}, {1}", val1, val2);
      Console.WriteLine("Refs: {0}, {1}", ref1.Value, ref2.Value);
      Console.WriteLine("Press ENTER to end");
      Console.Read();
    }
  }
}
```

Rezultat:



Rezimirajmo:

Promenljiva referencnog tipa pravi se svaki put kada deklarišete promenljivu kao:

- klasu
- interfejs
- niz
- string
- objekat
- delegat

Promenljiva vrednosnog tipa se pravi kada deklarišete promenljivu tipa:

- Integer
- Floating
- Boolean
- Enumeration
- Structure

Svojstva u C#

Svojstvo daje (postavlja) informacije o objektu kome pripada. Svojstvo je jedan metod ili par metoda (što se klijentskog koda tiče), ali se

ponaša kao polje. Na primer, kada pomerate neku kontrolu svojstvo *Location* daje vam informaciju o poziciji gde se ta kontrola nalazi. Tekstualni sadržaj je opisan svojstvom *Text*, itd.

Svojstvo može da da informaciju o objektu i/ili da postavlja neku vrednost za objekat. Drugim rečima svojstva mogu da budu *read-write,* ali i samo jedno od ta dva, bilo koje.

Svojstva su neobična i po tome što su preuzeta iz programskog jezika VB, a ne iz C++ ili Jave. Svojstva se ponašaju kao atributi, ali za razliku od njih svojstva ne predstavljaju samo vrednost neke promenljive, već su po prirodi funkcije kao i metode.

Na primer, upotreba jednog svojstva definisanog u klasi *Form*:

```
//frm1 - objekat ( tacnije forma ) i definisan je klasom System.Windows.Form
frm1.Height = 500;
```

Prilikom izvršenja ovog koda prozor će dobiti visinu 500, a korisnik će moći da vidi ovu promenu na ekranu. Sintaksno, kôd je isti kao da smo izvršili podešavanje nekog polja, ali suštinska razlika je u tome što je pozvan metod koji vrši promenu visine.

Sintaksa za definisanje jednog svojstva najbolje se vidi iz sledećeg primera:

```
public string myProperty
{
   get
   {
      // podesvanja
      return "Ovo je samo primer!";
   }
   set
   {
      // podesavanja
   }
}
```

Svojstvo se definiše u okviru neke klase kojom se opisuje objekat. U gornjem primeru svojstvo je tipa *string* i nosi naziv *myProperty*. Drugim rečima, vrši se čitanje i podešavanje vredosti nekog stringa uz sve ostale akcije koje su skrivene iza metoda koje definišu svojstvo.

Ukoliko svojstvo ima metode *get* i *set* za njega kažemo da je *read-write*. Ukoliko ima samo metodu *get* kažemo da je *read-only*, a ako ima samo metodu *set*, kažemo da je *write-only*.

Programski jezik C# ne dozvoljava podešavanje različitih modifikatora pristupa (*private, public* ili *protected*) pristupnim metodama *get* i *set.*

Napomena:

Čitaoca molimo da pogleda priručnike za OOP ili C++ ukoliko nije siguran u značenje prethodno pomenutih modifikatora pristupa.

Primer: U klasi *Property* dodato je svojstvo *Name*, a u klasi *Class1* pokazana je upotreba tog svojstva.

```
using System;

namespace ConsoleApplication1
{
   public class Property
   {
      public string Name     // Svojstvo (Properitet)
      {
         get
         {
            return name;
         }
         set
         {
            name = value;
         }
      }
      private string name;   // Field/Polje
  }

   class Class1
   {
      static void Main()
      {
         Console.Write("Name: ");
         Property property = new Property();

         property.Name = Console.ReadLine();          // set prop
         Console.WriteLine(property.Name);            // get prop
      }
   }
}
```

Dodavanje svojstava iz IDE

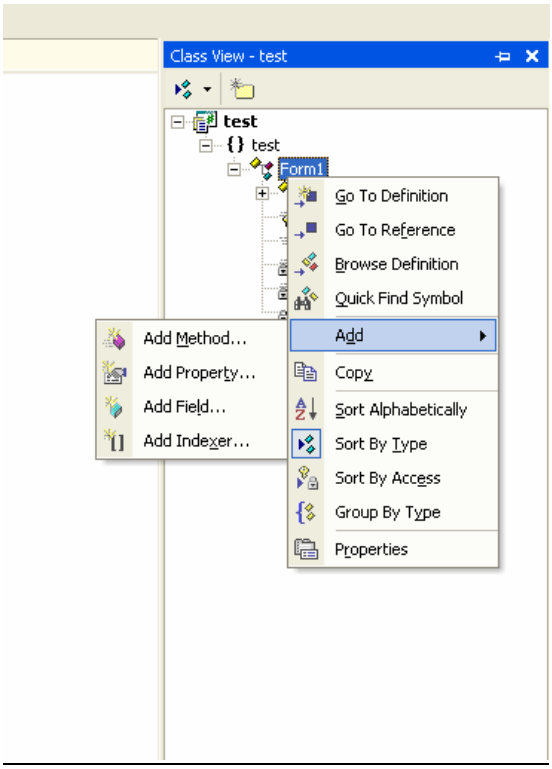
Osim što se mogu programski dodati na način koji je prikazan kroz primer, IDE daje sopstveni alat pomoću koga možete dodati proizvoljno

svojstvo nekoj klasi. Pretpostavimo da ste kreirali neku aplikaciju na način:

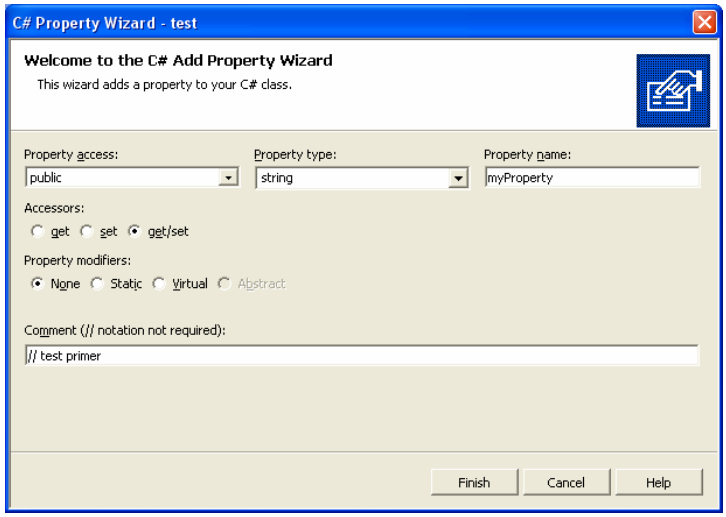
1. Pokrenite Microsoft Visual Studio .NET
2. Na stranici Start Page izaberite ***New Project*** (ili, na glavnom meniju, ***File->New->Project***...)
3. Na New Project dijalogu u ***Project Types*** listi birate ***Visual C# Projects***
4. U Templates listi birate ***Windows Application (.Net)***
5. U Name tekst polju zamenite <Enter name> sa test
6. U kombo polju Location prihvatite predlog ili otkucajte sami
7. ***OK***
8. Pritisnite F5 da biste testirali program
9. Zatvorite ga i vraćate se u VS

Ovo je standardni deo za pravljenje okvira jedne windows aplikacije u kome dalje delujemo.

10. Otvorite pogled na klase u vašoj aplikaciji Class View
11. Desnim tasterom miša kliknite na klasu Form1 u koju dodajemo novi metod
12. Pošto se otvara pomoćni meni, kao na slici, odaberite stavku ***Add***



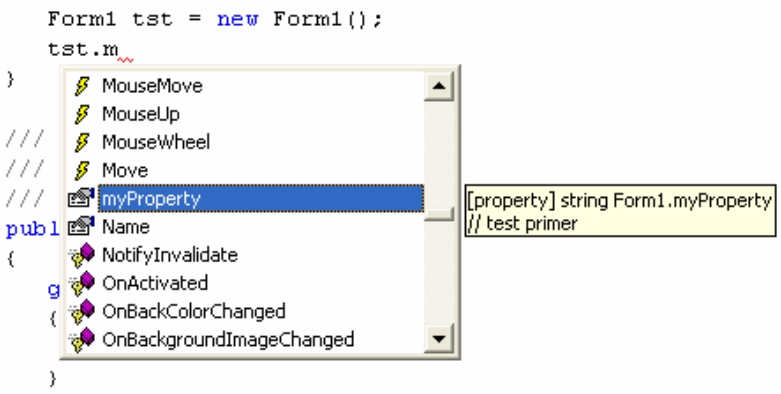
13. A zatim odaberite ***Add Property***
14. Otvara se novi dijalog u kome podešavate parametre koji se tiču svojstva koje dodajete, na primer kao na slici ispod



15. Pošto ste sve popunili završavate dugmetom Finish. Automatski se vraćate u kôd na mesto gde je kreirano telo novog svojstva zajedno sa komentarom.

```
/// <summary>
/// // test primer
/// </summary>
public string myProperty
{
   get
   {
     return null;
   }
   set
   {
   }
}
```

Pošto smo dodali naše svojstvo nekoj klasi ono postaje za IDE potpuno ravnopravno kao i sva ostala svojstva. Nadalje ovo svojstvo se nalazi u listi ponuđenih komandi koje nudi IDE pri pisanju koda. Pri tome je pravilno pridružena sličica koja označava da je reč o svojstvu, a dodatni žuti pravougaonik sa upustvom predstavlja onaj komentar koji ide uz svojstvo.



Za IDE editor se kaže da ima osobinu ***Intelli-Sense***. Kada počnete da kucate prikazuje imena klasa, polja, metoda, svojstava. Takođe, prikazuje listu parametara metoda kada se odabere ime metode. Ukoliko postoje preopterećene metode (mada je pojam poznat iz C++ objasnićemo ga kasnije) prikazuje listu metoda sa odgovarajućim parametrima.

3. DIZAJN KONTROLA

Kada kreirate neku aplikaciju koja sadrži interakciju sa korisnikom (teško je zamisliti primer bez toga) sasvim sigurno je da ćete morati da imate objekte kojima obezbeđujete razmenu informacija između aplikacije i korisnika. Ti objekti će prikazivati podatke korisniku ili će prenositi podatke od korisnika do vašeg programa. Ovi objekti se nazivaju kontrole (i to Windows kontrole).

Osobine Windows kontrola

Kontrole koje se postavljaju na forme po svojoj prirodi i nameni različite su. Istovremeno, postoje i mnoge sličnosti među njima koje su veoma značajne za programere. Prvo, kada pominjemo kontrole u ovom poglavlju mislimo na kategoriju Windows kontrola. Ovo je napomena koju autori namerno ističu zbog raznovrsnosti kontrola koje ovo okruženje nudi korisniku – tj. pogramеru.

Zajedničko za sve kontrole je da su po svojoj prirodi takođe i prozori. Šta to znači videćemo kasnije. Za sada recimo to da sve imaju većinu sličnih svojstava. Prihvatanje određenih događaja tipično je za kontrole. Neki događaji su specifični samo za određenu kontrolu dok su drugi zajednički.

Veoma je dobro i važno za programere da koriste određenu nomenklaturu kada je reč o dodeljivanju imena kontrolama, formama pa i tekućim promenljivima. Ime treba da bude što više opisno tj. da što više opisuje ono što predstavlja. Nekada je za to potrebno više od jedne reči. U nepisanoj C# nomenklaturi takva imena počinju malim slovom a svaka nova reč velikim. Na primer: firstName, dateOfBirth… Ukoliko ste nekada pisali duže programe jasno vam je o čemu se radi. Teško je održavati veći kôd bez ovakvih pravila. Ukoliko pak radite u timu razlog je još veći. U tom slučaju morate ulagati dodatan napor da razumete tuđi kôd, ali ako je on sličan vašem i ako postoje ovakva pravila to predstavlja manji problem.

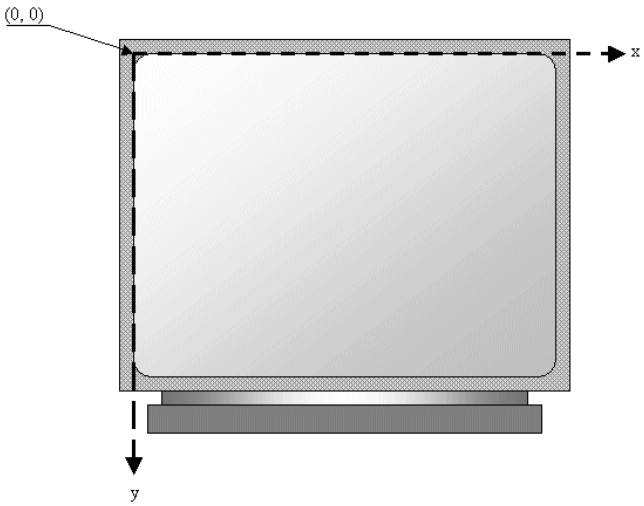
Relacija *roditelj-dete*

Postoji nekoliko različitih relacija između prozora koji čine jednu ili više aplikacija. Za nas u ovom trenutku, kada govorimo o pozicijama kontrola, bitan odnos je roditelj-dete tzv. *parent-child relationship*. Na primer, kontrola koju ste postavili na neku formu predstavlja prozor čiji roditelj je drugi prozor tj. *Forma*. Ovo je podrazumevana relacija kada se kontrole postavljaju i manifestuje se "lepljenjem" kontrole za tu formu. Pomeranjem forme pomeraju se sve kontrole na njoj, tj. sva deca prozori. Ista relacija postoji između formi koje pripadaju nakoj MDI aplikaciji. (Pojam MDI aplikacija biće kasnije detaljnije razmatran, za sada je dovoljno da pomenem kao primer "Word" ili "Exel" gde se istovremeno mogu otvoriti nekoliko dokumenata u posebnim prozorima i svi prozori su u zajedničkom prozoru glavne aplikacije.)

(Osim odnosa između prozora roditelj-dete postoji i druga relacija, koja bi na ovom mestu unela konfuziju, a ne bi doprinela razumevanju i nju autori svesno preskaču.)

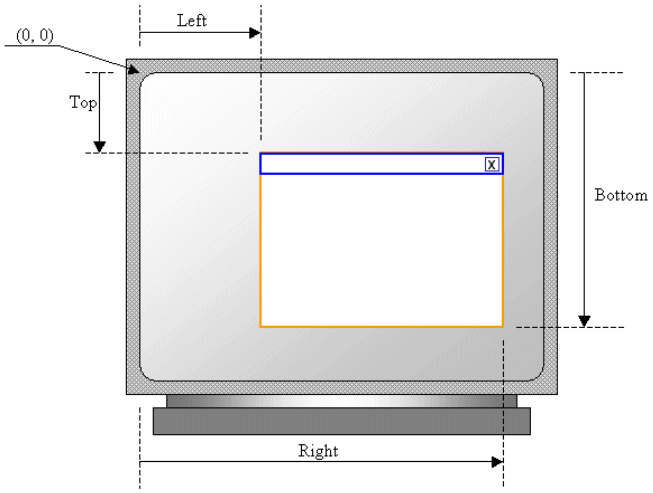
Koordinatni sistem

Da bi bolje razumeli svojstvo *Location* daćemo neka dodatna objašnjenja u vezi koordinatnog sistema koji se primenjuje kada je reč o postavljanju objekata na ekran i odnosima između prozora u Windows operativnom sistemu.



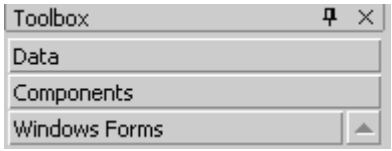
Pozicija nekog prozora čiji je roditelj desktop računa se od gornjeg levog ugla ekrana, kao što je to prikazano na slici. Ose koordinatnog sistema orijentisane su tako da rastu od leve strane nadesno za x osu i od vrha nadole za y osu.

Rastojanje od koordinatnog početka tačke do leve ivice je svojstvo "Left". Adekvatna ostala svojstva zovu se "Right", "Top", "Bottom".

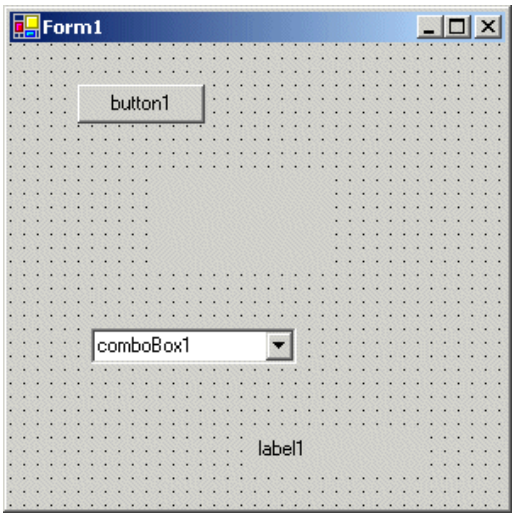


Postavljanje kontrole na formu

Kontrola se bira sa *toolbox* palete na levoj strani ekrana. Kategorije na paleti *toolbox* obezbeđuju nam brži izbor tražene kontrole.



Kada se željena kontrola nađe ona se mišem selektuje, a zatim "prebaci" na tekući (*current-važeći, tekući*) prozor (formu) takođe pritiskom miša. Kontrola će se naći na poziciji gde je bio kursor miša. Izgled jedne forme sa nekim kontrolama dat je ispod.

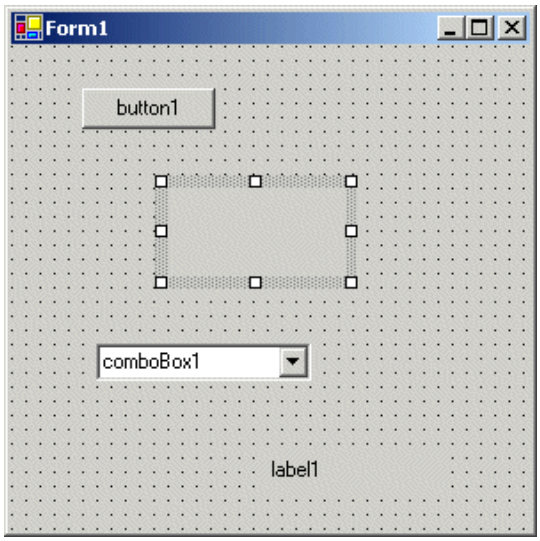


Postoje još neke varijante dodavanja kontrola na formu.

- Ukoliko uradite dvostruki klik na neku kontrolu ona će biti automatski dodata formi u gornjem levom uglu.
- Ukoliko morate da postavite puno istih kontrola na formu (što nije tako redak slučaj) pokušajte se tasterom Ctrl. Naime, selektujte kontrolu ali sa pritisnutim Ctrl tasterom (toolbox treba da bude fiksiran). Kontrola ostaje selektovana pa možete običnim klikom na miša da postavite ovu kontrolu proizvoljan broj puta na formu.

Selektovanje kontrole, podešavanje pozicije i veličine

Da biste mogli da vršite naknadne promene pozicije kontrole ili nekih svojstava, kontrola se mora selektovati. Ovo se izvodi jednostavnim klikom miša na nju. Oko kontrole se pojavljuje pravougaonik sa 8 malih kvadratića koji nam olakšavaju promene dimenzija kontrole u određenim pravcima.



Da bismo selektovali više od jedne kontrole moramo držati pritisnute tastere Shift ili Ctrl. Zatim se pritiska jedna po jedna željena kontrola. Poslednja ima pravougaonik sa crnim kvadratićima oko nje. Drugi način da se selektuje skup kontrola je pritiskom na površinu forme i povlačenjem miša se formira pravougaonik, a sve controle u njemu da postaju selektovane.

Pomeranje određene kontrole vršite tako što mišem "pritisnete" tu kontrolu a zatim dok držite miš pritisnut povlačite kontrolu preko forme. Pošto je pomerite na željenu poziciju otpustite taster miša i kontrola zauzima novu poziciju. Ako ste selektovali više od jedne kontrole na isti način možete pomerati i grupu kontrola.

Drugi način je da selektujete kontrolu a onda koristeći "strelice" na tastaturi izvodite pomeranje kontrole po formi. Korak pomeraja odgovara mreži na formi (*grid)*. Ukoliko želite preciznije pomeranje koristite pritisnut taster Ctrl.

Zajednička svojstva Windows kontrola

Name

Svaka kontrola ima svojstvo ***Name***. Ovo svojstvo omogućava da se preko imena pristupi nekoj kontroli ili formi. Ime mora postojati i ono

se automatski dodaje pri kreiranju, tj. pri postavljanju neke kontrole/forme. Na primer, kada se neka aplikacija formira automatski se postavlja prva forma i njoj se dodeljuje ime Form1. Svaka sledeća forma dobija novo ime sa narednim brojem: Form1, Form2....

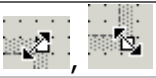
Isto pravilo važi i za kontrole koje se postavljaju na forme. One dobijaju imena po automatizmu koje im IDE okruženje dodeljuje a bazira se na tipu kontrole. Tako na primer tekst polja dobijaju imena *text1*, *text2*...

Location

Drugi primer uobičajenog svojstva svih kontrola je njihova lokacija (***Location***). Naravno ovo svojstvo je veoma bitno za vizuelni izgled forme, ali moram pomenuti da postoje kontrole koje se postavljaju na formu sa namerom da obezbede dodatnu funkcionalnost bez potrebe da budu vidljive. To je izuzetak kada pozicija kontrole nije od značaja. Ovo svojstvo je opisano koordinatama X i Y. Ovo je pozicija gornjeg levog ugla kontrole u odnosu na prozor na kome se kontrola nalazi. Obratite pažnju da je reč o vrednosnom tipu, jer su X i Y definisani kao celi brojevi) a celi brojevi se realizuju kao strukture.

Size

Kada je reč o grafičkim svojstvima kontrole, osim pozicije, treba spomenuti i njenu veličinu ***Size***. Svi objekti su po prirodi pravougaonog oblika pa se veličina jednostavno opisuje. Svojstva koja opisuju veličinu su: širina (*Width*), odnosno visina (*Height*). Ukoliko se kontrola ne postavlja programski već u toku dizajna forme, promena veličine se obavlja jednostavnim korišćenjem miša, uz pomoć pravougaonika koji označava selektovanu kontrolu. Na pravougaoniku se mišem pritisne kvadratić zavisno od dimenzije koju želimo da promenimo, a zatim se povlačenjem miša menja dimenzija kontrole.

Cursor	Funkcija
	Promena visine
	Promena širine
	Promena i visine i širine u različitim pravcima

Drugi način da se promene parametri kontrole (ne samo veličina) je koristeći Properties prozor. Među svojstvima kontrole lako se prepoznaje svojstvo Size ispred koga se nalazi znak +. Ovaj znak, tipično za Windows, omogućuje ekspandovanje stavke. Ispod ove stavke se nalaze posebno Width i Height.

Naravno, sva svojstva mogu se postavljati i programski. Bilo na samom startu pri kreiranju kontrola/formi, bilo u kodu.

Na primer:

```
static void Main()
{
   Form1 Fm = new Form1();
   Fm.Size     = new System.Drawing.Size(450, 320);
   Application.Run(Fm);
}
```

Bounds

Pozicija zajedno sa veličinom smeštene su u jednom svojstvu kontrole koje se naziva **Bounds**. Ovo svojstvo daje pravougaonik kontrole/forme. U kodu ovim svojstvom lako se vrši repozicioniranje i promena veličine kontrole.

Text

Veoma korisno svojstvo svake kontrole/forme je **Text**. Naravno da postoji kod kontrola koje mogu prikazati tekst. Na primer, kod kontrole *labela Text* svojstvo čuva tekst koji kontrola prikazuje. Kod kontrole dugme predstavlja tekst koji se prikazuje na dugmetu. Dalje, kod forme predstavlja tekstualni sadržaj u naslovnoj liniji itd, itd.

Promena ovog svojstva izvodi se u toku dizajna koristeći Properties prozor i biranjem Text polja a zatim ukucavanjem sadržaja. Obratite pažnju da je podrazumevani tekst u **Unicode** formatu, tj. lako možete koristiti srpska ili neka druga slova. Naravno da se tekst može menjati i programski. Na primer:

```
static void Main()
{
   Form1 Fm = new Form1();
   Fm.Text     = "Employment Application";
```

```
    Fm.Size     = new System.Drawing.Size(450, 320);
    Application.Run(Fm);
}
```

Visible

Ranije smo pomenuli svojstvo koje određuje da li je kontrola vidljiva ili ne. Ovo svojstvo se naziva **Visible** a vrednosti su bulovog tipa (Boolean type) **True/False**. Ako kontrola nije vidljiva onda se uobičajeno kaže da je sakrivena - hidden. Ovo svojstvo takodje se može programski menjati jednostavnim dodeljivanjem vrednosti. Ukoliko kontrola nije vidljiva onda nije ni dostupna – logično. Drugim rečima ona je za korisnika sa stanovišta unosa i prikaza informacija potpuno bez funkcije. To ne znači da su takve kontrole nepotrebne – naprotiv. U toku izvršavanja programa one se mogu učiniti vidljivim i dostupnim, ili nevidljivim i nedostupnim, u zavisnosti od toga kakva vrsta podataka se unosi i šta se sa kontrolom radi.

Enabled

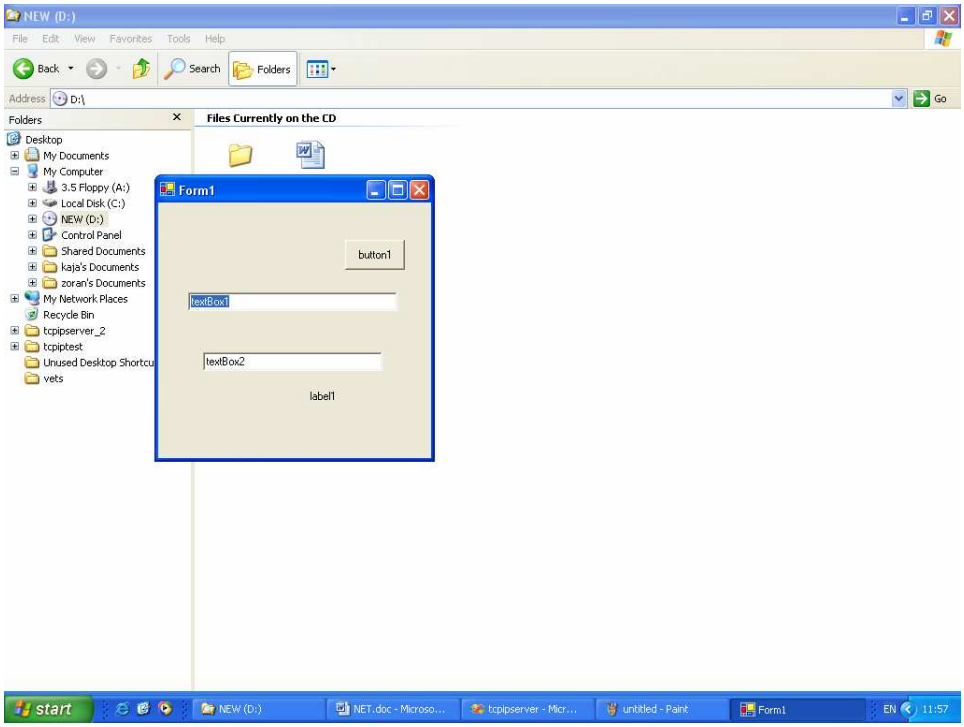
Kada je kontrola vidljiva to ne znači obavezno da je i dostupna korisniku. Posebno svojstvo definiše da li kontrola može primiti ulaz od korisnika čak i kada je vidljiva. Ovo svojstvo se naziva **Enabled** i takođe je bulovog tipa. Ako je kontrola vidljiva, a *Enabled* svojstvo postavljeno na vrednost *false*, to znači da korisnik može samo čitati vrednosti te kontrole bez mogućnosti da ih menja. Podrazumevani izgled kontrole u tom slučaju je blago zasivljenje (*dimmed*). Ovo svojstvo se često programski, u samom kodu menja. Obično kada unos nekih podataka povremeno (ili stalno) treba zabraniti, a prikaz podataka ostaviti.

Fokus kontrole i aktivan prozor

Pojam fokusa i aktivnog prozora bitni su za razumevanje relacija među prozorima i za objašnjavanje nekih svojstava (a kasnije i događaja).

Za kontrolu se kaže da je u fokusu ako kontrola može da primi ulaz sa tastature. Takvo stanje poseduju kontrole, ali na primer i stavke u listi. Ono se može posebno grafički označiti što se u podrazumevanom vizuelnom podešavanju i radi.

Pojam aktivnog prozora ne vezuje se za kontrolu koja pripada nekoj formi već za samu formu. Kontrola koja ima fokus nalazi se na aktivnom prozoru. Samo jedan prozor je aktivan u jednom trenutku.



Na gornjoj slici dat je prikaz jednog primera. Jasno je da je Form1 aktivni prozor a textBox1 u fokusu i da se nalazi na tom prozoru.

TabIndex i *TabStop*

Ovde ćemo pomenuti još jedno veoma bitno svojstvo u programiranju na grafičkim okruženjima (GUI). Ovo svojstvo nije vidljivo – bar ne na prvi pogled. Obično ga nazivamo tab redosled ili u originalu ***tab ordering***. Kretanje sa kontrole na kontrolu na jednoj formi koja je aktivna možete izvesti i preko tastature koristeći Tab taster. Kada se on pritisne fokus kontrole se prebacuje na sledeću. Upravo tab svojstvo određuje redosled kontrola u ovom slučaju.

I ne samo to. Kada govorimo o tab redosledu i tab svojstvima treba znati jos nešto. Postoje kontrole koje ne mogu primiti fokus – na primer labela. Takođe, kontrola koja može da primi fokus mora imati svojsto ***TabStop*** postavljeno na *true,* ako treba da primi fokus. U suprotnom ta

kontrola će biti preskočena dok pritiskate taster Tab. Na sreću, ovo svojstvo je za kontrole koje mogu primati fokus inicijalno postavljeno na *true*. Kontrole koje ne mogu primiti fokus (primer Labela) nemaju ovo svojstvo.

Koja pozicija u tab redosledu pripada određenoj kontroli definiše svojstvo ***TabIndex***. Ukoliko tu poziciju želite da promenite treba da promenite vrednost ovog svojstva.

Programsko kreiranje kontrola

Već smo videli kako se kreiranje kontrola izvodi na vizuelan način, selektovanjem određenih kontrola sa liste *Toolbox* i dodavanjem na formu. Postoje razlozi kada se kreiranje kontrole ne može izvesti na taj način već morate to uraditi programski.

Klase za upravljanje kontrolama nalaze se u prostoru imena **System**. Unutar ovog prostora imena nalazi se **Windows,** a u njemu prostor imena **Forms**. Sve kontrole raspoložive za .Net programiranje su kreirane u **System.Windows.Forms** prostoru imena.

Svaka kontrola predstavlja jedan tip objekata (promenljive). Kontrole su opisane klasama pa se deklarisanje kontrola obavlja na isti način kao i svih ostalih promenljivih.

ClassName VariableName;

Na primer: Deklarisanje objekta *btnSubmit* tipa *Button* u prostoru imena *System.Windows.Forms.*

System.Windows.Forms.Button btnSubmit;

Da bi kontrola (objekat) zaista mogla da se koristi mora se odvojiti i memorijski prostor, korišćenjem operatora **new**, i odraditi sva neophodna podešavanja. Podešavanja se obavljaju u konstruktoru koji se automatski poziva pri kreiranju objekta pozivom operatora *new*.

Dat je primer "ručnog" dodavanja kontrole u vašu klasu Form1.

```
public class Form1 : System.Windows.Forms.Form
{
      System.Windows.Forms.Button btnSubmit;
      /// <summary>
      /// Required designer variable.
      /// </summary>
      private System.ComponentModel.Container components =
null;

      public Form1()
      {
            btnSubmit = new System.Windows.Forms.Button();
            btnSubmit.Location = new
System.Drawing.Point(88, 2);
            btnSubmit.TabIndex = 0;
            btnSubmit.Text = "Submit";
            Controls.Add(btnSubmit);

 ...

}
```

Linijom koda

```
      System.Windows.Forms.Button btnSubmit;
```

na početku se deklariše promenljiva *btnSubmit*. Deklarisana je kao podatak koji pripada klasi *Button* u okviru prostora imena *Forms*, tačnije: *System.Windows.Forms.Button*. Pošto je reč o referentnom tipu podataka moramo odvojiti memorijski prostor na *heapu* za ovaj objekat i pozvati konstruktora tog objekta.

Ovo se obavlja preko operatora new:

```
      btnSubmit = new System.Windows.Forms.Button();
```

Sa ove dve linije koda mi jesmo kreirali objekat u osnovnom programskom smislu, ali ukoliko prevedemo kôd i startujemo program videćemo da se dugme ne nalazi na formi tj. da kreiranje nije izvedeno tako da bude i vidljivo.

Inicijalna vrednost svojstva Visible je true tako da nije neophodno podešavanje ove vrednosti. Takođe, svojstvo Size ima početnu vrednost.

Ostaje da podesimo poziciju dugmeta i opciono tekst na dugmetu. Pozicija je vrednosnog tipa i podešavanje izvodimo postavljanjem vrednosti direktno ili korišćenjem objekta *Point(,)*. U ovom drugom slučaju koji je dat u primeru objekat *Point* mora se kreirati operatorom *new*.

```
btnSubmit.Location = new System.Drawing.Point(88, 2);
```

Naredbe koje postavljaju tekst na dugme i Tab redosled nisu neophodne za kreiranje dugmeta na formi.

```
btnSubmit.TabIndex = 0;
btnSubmit.Text = "Submit";
```

Ono što je neophodno da bi dugme zaista zaživelo na formi je naredba:

```
Controls.Add(btnSubmit);
```

Koristi se svojstvo *Controls* i metod *Add*. Svojstvo *Controls* je *read-only* tipa i može se koristiti samo za dobijanje objekta *Control.ControlCollection* koji sadrži kolekciju kontrola (koje pripadaju nekoj kontroli – Formi). *Add* je metod koji pripada ovoj klasi i obezbeđuje dodavanje novog objekta (tipa Control) kontejneru.

Uporedimo ovaj kôd sa onim što se događa kada jednu kontrolu sa *toolbox*-a postavimo na formu. Prvo dodajmo jedno dugme sa svim podrazumevanim vrednostima na neku poziciju na formi. Ova akcija je adekvatna sledećim promenama u kodu u klasi koja opisuje formu na koju se postavlja kontrola:

1. Dodata je deklaracija objekta koji predstavlja tu kontrolu. U našem slučaju prvo je dodato jedno dugme:

```
private System.Windows.Forms.Button button1;
```

2. Unutar funkcije InitializeComponent() vrši se kreiranje objekta, a zatim i njegova inicijalizacija.

```
      this.button1 = new System.Windows.Forms.Button();
      //
      // button1
      //
      this.button1.Location = new System.Drawing.Point(96,
120);
      this.button1.Name = "button1";
      this.button1.Size = new System.Drawing.Size(136, 24);
      this.button1.TabIndex = 0;
      this.button1.Text = "button1";
```

3. U istoj metodi se kasnije vrši dodavanje ove kontrole kontejneru *Controls* u formi

```
        //
        // Form1
        //
        this.Controls.AddRange(new
System.Windows.Forms.Control[]{this.button1});
```

Primetite da dizajner koristi metod *AddRange*(…), a ne *Add*(…), koji je gore korišćen za ručno dodavanje jedne kontrole u kontejner.

Dakle iza jednostavne tehnike *drug and drop* stoji desetak linija koda. Bitno je da razumete da je sve što uradite koristeći IDE predstavljeno promenama u kodu projekta na kome radite. Drugim rečima, sve je to moguće i programski na isti, ili sličan, način uraditi. Alati vam stoje na raspolaganju za jednostavniji i lakkši rad.

Sada ćete videti šta se događa ako dodate formi još jedno dugme i jednu labelu, na isti način (preko liste *toolbox*).

1. Deklaracije

```
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.Label label1;
```

2. Kreiranje i inicijalizacija

```
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.button2 = new System.Windows.Forms.Button();
    this.label1 = new System.Windows.Forms.Label();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(96, 120);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(136, 24);
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
    //
    // button2
    //
    this.button2.Location = new System.Drawing.Point(152, 168);
    this.button2.Name = "button2";
```

```
        this.button2.Size = new System.Drawing.Size(88, 24);
        this.button2.TabIndex = 1;
        this.button2.Text = "button2";
        //
        // label1
        //
        this.label1.Location = new System.Drawing.Point(112, 64);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(120, 24);
        this.label1.TabIndex = 2;
        this.label1.Text = "label1";
        //
        // Form1
        //
        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
        this.ClientSize = new System.Drawing.Size(292, 266);
        this.Controls.AddRange(new System.Windows.Forms.Control[] {
                                                    this.label1,
                                                    this.button2,
                                                    this.button1});
        this.Name = "Form1";
        this.Text = "Form1";
        this.ResumeLayout(false);

    }
```

Postavljanje kontrole u fokus

Ranije smo objasnili pojam fokusa kontrole u Windows operativnim sistemima. Naglasimo još jednom da fokus označava da je kontrola spremna da primi ulazne podatke. Na primer, ako se neko dugme nalazi u fokusu to se prikazuje tačkastim pravougaonikom oko teksta na tom dugmetu (obratite pažnju na oznaku za podrazumevano dugme jer se ona često meša sa fokusom). Ako se tekst kontrola nalazi u fokusu onda se u njoj nalazi kursor koji trepće (tada je očigledno da kontrola prima ulaz sa tastature). Kada se neka stavka liste nalazi u fokusu tada je ona oivičena tačkastim pravougaonikom.

Programski fokus dajemo nekoj kontroli pozivom metode **Focus()**. Na primer:

```
    bool Focus();
```

Zadatak: Dodajte formi dve tekst kontrole upotrebom Dizajnera i podesite da druga kontrola ima inicijalno fokus.

4. DOGAĐAJI

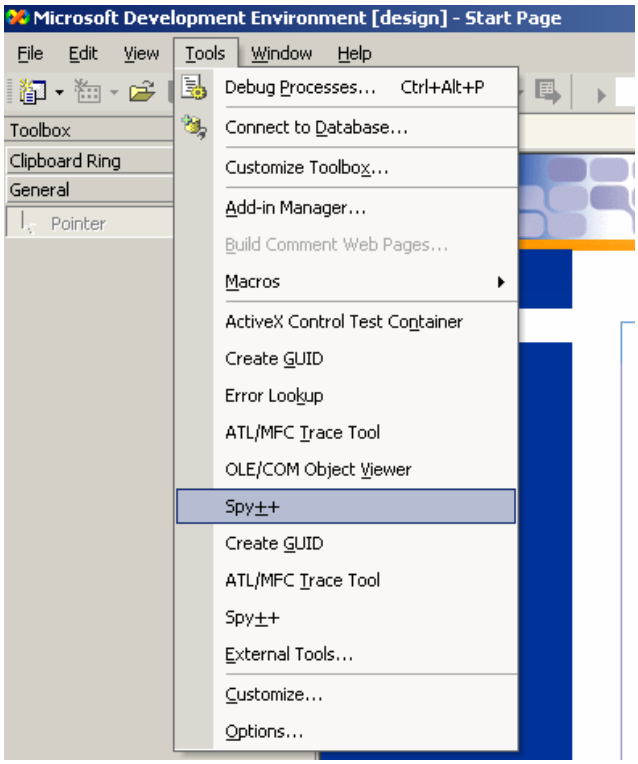
Događaji (*events*) obezbeđuju aplikacijama da prepoznaju i da reaguju na akcije bilo korisnika bilo drugih aplikacija. Pripadaju najvažnijim osobinama višeprocesnih operativnih sistema kao što je Windows. U toku rada neke aplikacije kontrole i forme (preciznije - prozori) neprekidno šalju poruke operativnom sistemu o svim promenama i akcijama korisnika. Zamislite vašu radnu površinu na računaru na kojoj je obično pokrenuto nekoliko aplikacija. Dodajte svakoj aplikaciji prozore koji joj pripadaju, na kraju dodajte veliki broj servisa koji rade u pozadini i koje ne vidite. Jasno vam je da je broj poruka koji se obradi u jednoj sekundi jako veliki i da ne sme da optereti operativni sistem. Višeprocesni rad baziran je na događajima.

Iako različitih poruka ima jako puno postoje i poruke koje programeri sami mogu dodavati za svoje potrebe. Poruke su svrstane prema svome značenju ili tipu kontrole kojoj pripadaju. Od programera se očekuje da zna za osnovne poruke i da ume, koristeći *Help,* da pronađe sve informacije o nekoj poruci.

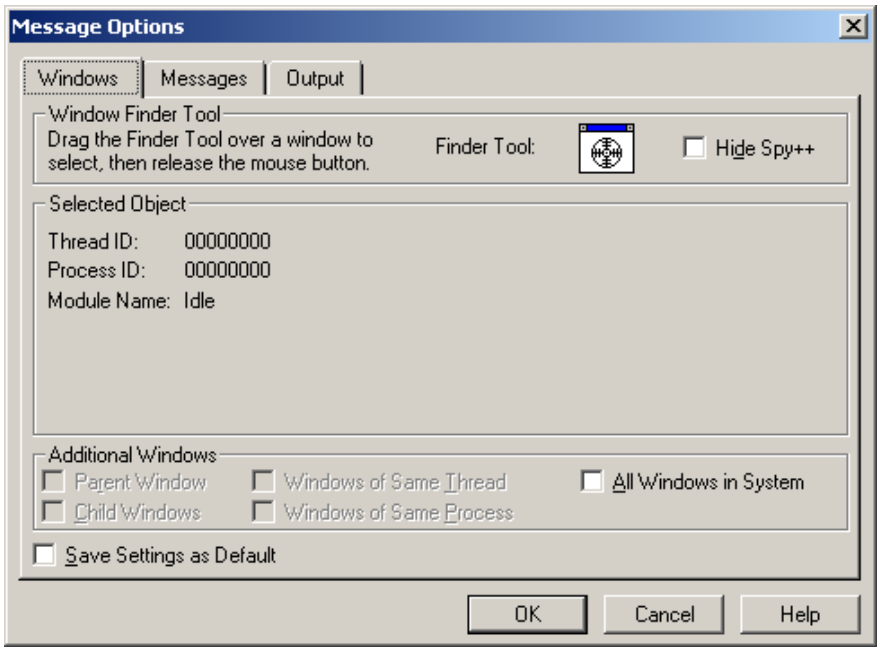
Spy++

Spy++ je poseban programski alat, deo *Visual Studia,* namenjen praćenju poruka nekog procesa ili prozora (i još ponešto). Spy++ možete da pokrenete iz menija *Tools* ili sa spiska programa

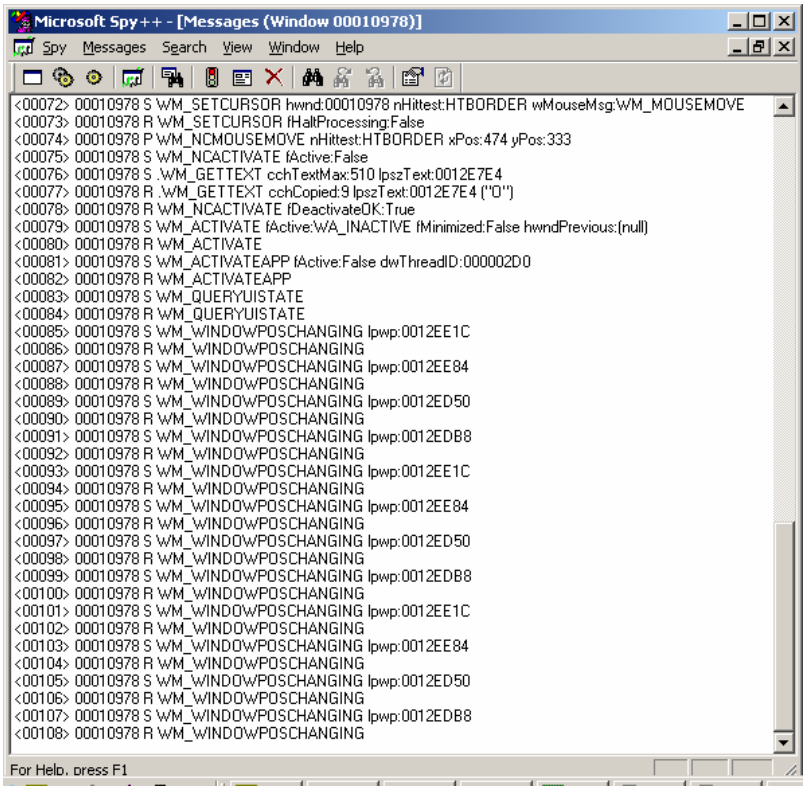
Programs->Microsoft Visual Studio .Net –> Visual Studio .Net Tools->Spy++



Sa menija odaberite *Spy->Log Messages...*



a zatim koristeći alatku *FinderTool* odaberite prozor koji želite da pratite. Karticom *Messages* podešavate, filtrirate, vrste poruka koje želite da pratite. Nadalje sve poruke koje se obrađuju u tom prozoru (a koje ste filtrirali) možete da pratite u prozoru, kao na slici



Još jednom, događaj je akcija kojom se poruka obrađuje, dok je poruka informacija koja se obrađuje. Dakle, događaj je opisan sa dva podatka: vrednost koju poruka nosi sa sobom i metod kojim se ta poruka obrađuje. Oba podatka su argumenti jednog događaja. Prvi mora biti klasa izvedena iz klase **EventArgs,** a drugi mora biti delegat.

Pre nego što nastavimo sa delegatima navedimo još par pojmova. Objekat koji izaziva (triggers) neki događaj naziva se *event sender* – pošiljalac događaja. Objekat koji hvata događaj naziva se *event receiver* ili primalac događaja.

Delegati

Delegat je pojam uveden sa C# (čitaj .NET). Delegati se koriste za prenos metoda kao argumenata drugim metodama. Mada se delegati mogu smatrati novom vrstom objekata suštinski delegati su pokazivači

na funkcije, koje ste sretali u programskom jeziku C. Sa bitnom osobinom da su tipski bezbedni (*type-safe function pointer or a callback)*.

Kod svih objekata u programskom jeziku C# (pa ako hoćete i u C++) pri korišćenju neke klase/delegata prvo morate da deklarišete odgovarajući objekat koji je tipa te klase/delegata, a zatim da napravite primerak tog objekta.

Deklaracijom delegata opisuje se vrsta metode koju delegat predstavlja. Sintaksa jedne deklaracije delegata može da bude:

```
delegate void myDelegate(int a, int b);
```

Prethodni primer opisuje metod koji vraća *void* a kao parametre ima dve celobrojne vrednosti tipa *int*. Ako se prisetite programskog jezika C jasno ćete uočiti analogiju kada se koriste pokazivači na funkcije.

```
delegate int myDelegate(int a, int b);
```

je deklaracija delegata koji vraća celobrojnu vrednost, a za ulazne argumente ima dve celobrojne vrednosti. Pretpostavimo da negde u vašoj klasi imate definisane dve funkcije koje odgovaraju delegatu, tj. da vraćaju *int* i da imaju dva argumenta tipa *int*. Često se kaže da imaju isti potpis. Na primer:

```
int add( int a, int b ){return a+b;}
int sub( int a, int b ){return a-b;}
```

Kada pravimo analogiju sa objektima recimo da ove dve funkcije odgovaraju klasama za tipične objekte. One definišu ponašanje delegata koje tek treba da kreiramo. Kreiranje delegata vrši se na isti način kao i kod svih drugih objekata, pomoću operatora *new*.

```
myDelegate x = new myDelegate( add );
myDelegate y = new myDelegate( sub );
```

Nadalje, imamo dva objekta x,y koji su po svojoj prirodi funkcije, a koje možemo koristiti na isti način kao i te funkcije kojima su oni opisani, dakle:

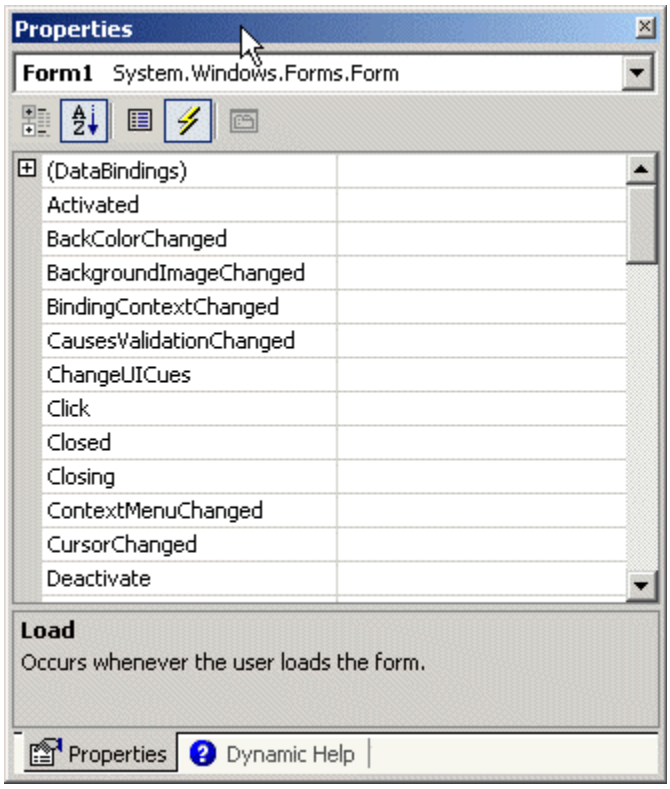
```
int r = x( 4, 3 ); // r = 7;
r = y( 4, 2 );     // r = 2;
```

Vežba: Napraviti program za sortiranje niza (koji će koristiti algoritam QuickSort ili BubbleSort), sa proizvoljnom funkcijom za poređenje dva elementa niza.

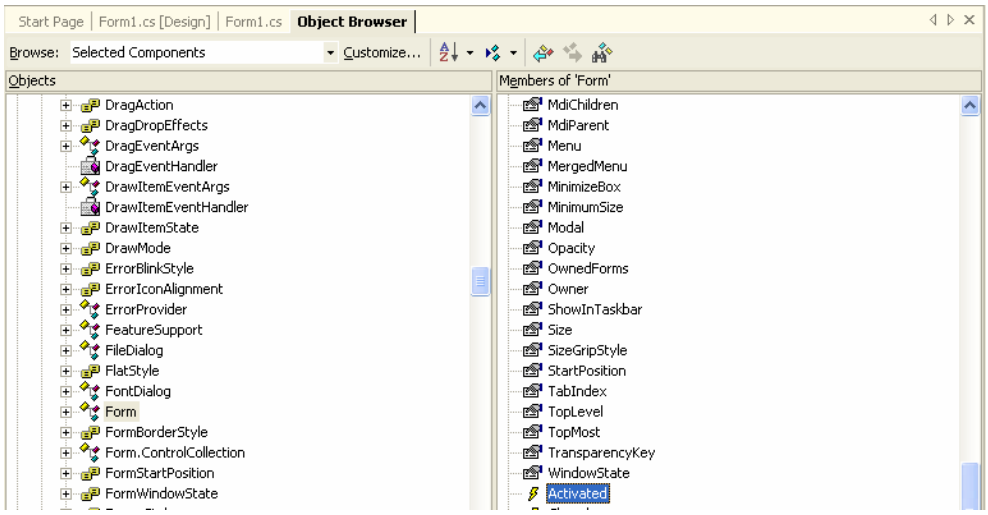
Pošto sada poznajete osnovne stvari o delegatima možemo da nastavimo započetu priču o događajima.

Događaji i IDE

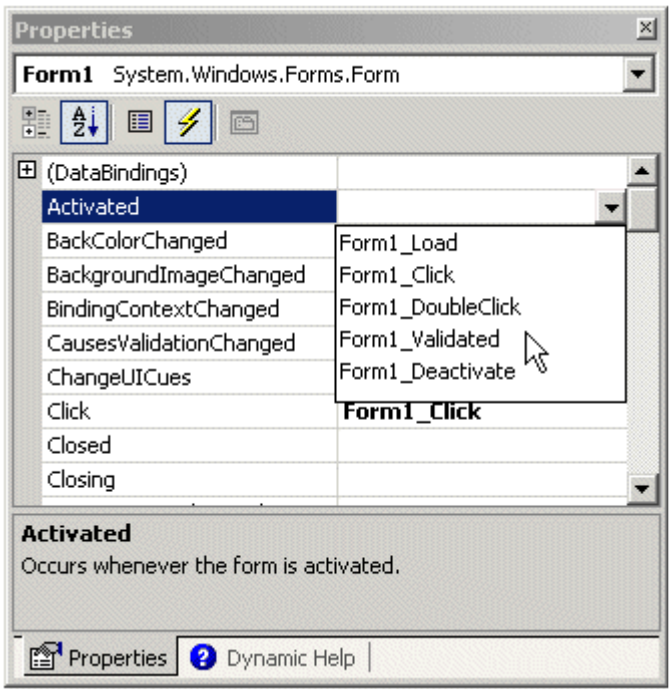
IDE daje programeru prilično jasnu sliku o događajima koje može da pridruži nekoj kontroli ili formi. Na prozoru *Properties* postoji dugme *Events*  . Kada kliknete na to dugme dobijate spisak događaja koji su pridruženi kontroli ili formi koja je selektovana. Na slici ispod dat je prikaz *Properties* prozora sa listom događaja koji stoje na raspolaganju korisniku za formu.



Događaje koji su vezani za neki objekat možemo videti i ako otvorimo *Object Browser*, kao na slici ispod:



Vratimo se na prozor Properties i otvorimo listu sa događajima. Lista je podeljena u dve kolone. Ime svakog događaja prikazano je na levoj strani. Kratak opis nalazi se na samom dnu i on vam može pomoći oko lakšeg snalaženja. (Taj opis se može isključiti ako smeta koristeći kontekstni meni sa tool bara.) Sa desne strane je kombo polje gde možete uneti novo ime i time kreirati novu metodu koja će biti vezana za taj događaj, ili izabrati neku od već postojećih sa liste.



Pretpostavimo da ste uneli novo ime *myHandler,* a zatim pritisnuli taster *Enter*, vraćate se u kôd na mestu ge je kreirana nova funkcija

```
private void myHandler(object sender, System.EventArgs e)

{

}
```

koja je zadužena za događaj *Activated.* Osim ove promene u kodu naći ćete u funkciji

```
private void InitializeComponent()
```

liniju koda kojom se nova funkcija pridružuje događaju *Activated*

```
this.Activated += new System.EventHandler(this.myHandler);
```

Zapazite da je metod *myHandler* proizvoljnog imena i da je zadužen za posebne korisničke akcije (*myHandler* nije jedina akcija koja se obavlja kada se registruje događaj) . Funkcija *myHandler* ima tačno definisane ulazne argumente (dve promenljive tipa *object* odnosno *EventArgs*) i povratnu vrednost tipa *void*.

Možda ste se upitali "gde su delegati"? Delegati koji se tiču standardnih događaja, tj. događaja koje šalje operativni sistem, već su definisani u okviru klasa koje opisuju Windows kontrole.

EventHandler je delegat definisan u .NET-u u prostoru imena *System* sa deklaracijom

```
public delegate void EventHandler(object sender, EventArgs e);
```

i predstavlja potpis svih funkcija za obradu događaja.

Operatorom *new* napravljen je pokazivač na funkciju koja je identična funkciji *myHandler* i taj pokazivač je dodeljen posebnom objektu *Activated* koji pripada klasi *Form*.

Zašto je *Activated* poseban objekat? Ako pogledamo u spisak svih članova klase *Form* naći ćemo i definiciju

```
public event EventHandler Activated;
```

Dakle, vidimo da postoji ključna reč **event** i ona definiše da li neki delegat odgovara "događaju", tj. nekoj promeni ili akciji. Delegat može imati jedan ili više pridruženih metoda.

Rezime: Da biste napisali svoje metode koje će reagovati na događaje morate proći kroz sledeće korake:

1. Kreiranje delegata. Ako definišete sopstveni događaj, morate osigurati da postoji delegat koji će se koristiti sa ključnom reči event. Ako je događaj predefinisan u .NET Framework-u dovoljno je da znate samo naziv delegata;
2. Kreiranje klase koja sadrži:
    a. događaj kreiran od nekog delegata,
    b. metode koje pozivaju (izazivaju) neki događaj;
3. Definišite jednu ili više klasa koje povezuju metode sa događajem. Svaka od ovih klasa sadrži:
    - pridruživanje jedne ili više metoda, koristeći += ili -= operatore, događajima u baznoj klasi,
    - definiciju metoda koje će biti pridružene događaju;
4. Korišćenje događaja:
    - kreirajte objekat klase koji sadrži deklaraciju događaja. ,
    - kreirajte objekat klase koji sadrži definiciju događaja.

Komanda dvostrukog klika (ili pritisak na taster enter) na neku stavku na nekoj aplikaciji izaziva podrazumevanu akciju. Ukoliko često pravite tipične komande možete se poslužiti skraćenim postupkom za kreranje podrazumevanog događaja neke kontrole. Kada se ta komanda izvede u IDE okruženju, nad nekom kontrolom u toku dizajna, kompajler kreira podrazumevani događaj. Vrsta podrazumevanog događaja zavisi od kontrole do kontrole.

Ako na formu postavite neko dugme sa podrazumevanim parametrima i ako pritisnete taster *Enter* dok je kontrola selektovana, IDE će formirati metod za rukovanje događajem *pritisak na levi taster miša (Click).* Evo šta se dogodilo u kodu:

1. Dodata je privatna metoda sa podrazumevanim imenom, koja će odraditi posebne akcije kada se izvede pritisak na miša. Pošto je ime dugmeta ostalo podrazumevano - *button1* - na isti način se formira i ime ove metode pošto se još doda naziv događaja. Dakle, dodata je funkcija

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

2. Zatim je događaju *Click,* koji je definisan u klasi Form, dodata nova funkcija za obradu koju smo pomenuli u tački 1., dakle

```
this.button1.Click+=new
System.EventHandler(this.button1_Click);
```

Primer: Napišite konzolnu aplikacju koja će periodično (na svakih 100 ms) da ispiše vrednost celobrojnog brojača svaki put u novoj liniji i uvećen za 1.

```
using System;
using System.Timers; //dodati obavezno

namespace timer
{
  class Class1
  {
    int brojac = 0;
    Timer myTimer;

    void inittimer()
    {
      myTimer = new Timer( 100 ); // definisan je period od 100ms

  /* metodama koje reaguju na dogadjaj kada istekne 100ms dodata
   * je fja myAction
   * potpis ovog metoda odgovara ElapsedEventHandler
   */
myTimer.Elapsed += new ElapsedEventHandler( myAction );
    }

    [STAThread]
        static void Main(string[] args)
        {
      // instanciranje objekta x tipa Class1
      Class1 x = new Class1();
      // inicijalizacija tajmera
      x.inittimer();
      // start tajmera
      x.myTimer.Start();
      // zatvaranje konzolnog prozora pritiskom na ENTER
      Console.ReadLine();
        }

void myAction( object source, System.Timers.ElapsedEventArgs e )
    {
      brojac ++;
      Console.WriteLine( brojac.ToString() );
      if( brojac >= 100 )
        myTimer.Stop();
    }
  }
}
```

Događaji vezani za forme i kontrole

Paint

Iscrtavanje kontrola u grafičkim okruženjima čini kontrole vidljivim i predstavlja veoma bitnu i zahtevnu operaciju. Kontrole se iscrtavaju kada se vrši promena njihovog položaja ili kada tek postaju vidljive. Kad god treba iscrtati kontrolu izvršava se događaj **Paint**. Sintaksa ovog događaja je:

```
void PaintEventHandler(object sender, PaintEventArgs e);
```

Znači, da bi se iscrtavanje izvelo, **PaintEventArgs** parametar se prosleđuje delegatu **PaintEventHandler**. Ovaj parametar nosi informaciju o oblasti koju treba iscrtati i grafičkom objektu za iscrtavanje.

Resize

Veličina kontrole je parametar koji se menja u toku izvršavanja aplikacije bilo akcijama minimizovanja, ili maksimizovanja prozora, bilo namernim podešavanjima koje je obezbeđeno kodom te aplikacije. Pri tome nije samo bitna nova pozicija i veličina kontrole već i prethodna. Razlog je što u ponašanju kontrole treba obezbediti da se ona može vratiti na prethodnu veličinu.

Kada se veličina kontrole promeni nastaje događaj ***Resize*()** sa tipom ***EventArgs***.

Događaji tastature

Posebna klasa događaja koje windows kontrole obrađuju su događaji sa tastature. Tastatura predstavlja jedan od dva osnovna uređaja za unos podataka u kontrole. Drugi je miš.

Kad god se pritisne neki taster aktivira se događaj ***KeyDown,*** a tip poruke je ***KeyEventArgs***. Analogno postoji i poruka kada se taster "otpusti" - ***KeyUp*** i ona je takođe tipa ***KeyEventArgs***. Ukoliko korisnika interesuje da detektuje koji karakter je pritisnut prethodne dve poruke nisu dobar izbor. ***KeyPress*** poruka se emituje ako korisnik pritisne taster koji je tipa karaktera. Dakle, ovo mora biti neki prepoznatljiv simbol. Mala slova, brojevi, znaci interpunkcije itd. Ukoliko se na primer pritisne Shift+5 tj. taster % detektuje se kao jedan karakter tj. KeyChar=37 '%'.

Događaji miša

U nameri da ne ponavljamo ono što svi znaju nećemo da objašnjavamo šta je miš i kako radi. Pomenućemo samo osnovne događaje vezane za ovaj uređaj koji služi za unos podataka.

Upotreba miša izaziva: promenu pozicije kursora, pritisak tastera (levo, desno, srednje), otpuštanje istih tastera i na kraju skrolovanje ako miš ima točkić. Takođe se posebno detektuje i dvosturki klik. Pritisak na levi taster je **Click**, a na desni **Right-Click**.

MouseDown je poruka koja se događa kada se pritisne levi taster miša. ***MouseEventArgs*** je tip ove poruke.

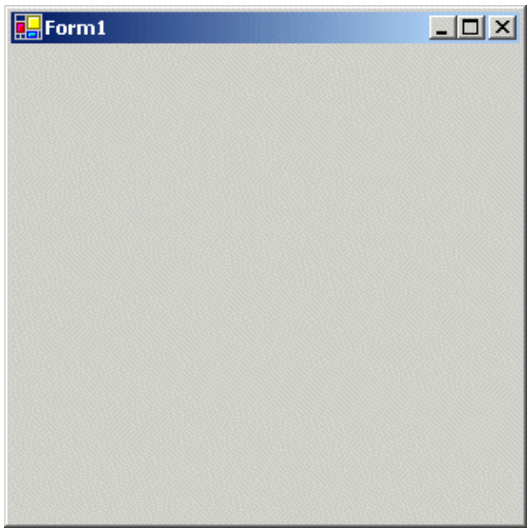
Analogno je i za otpuštanje tastera tj. ***MouseUp***, odnosno ***MouseEventArgs*** koji se prosleđuje ***MouseEventHandler*** na obradu.

Pomeraj miša se detektuje porukom ***MouseMove*** tipa ***MouseEventArgs***.

5. FORMA

Izgled forme

Forma je osnovni grafički objekat koji se koristi u aplikacijama koje rade u grafičkom okruženju kao što su Windows-i. Sama forma ne znači ništa. Njena osnovna uloga je da "drži" kontrole i obezbeđuje interakciju korisnika sa računarom.



Polazna (ili osnovna) forma dobija se na startu. Kreiranje neke druge obavlja se kasnije eksplicitno kroz dizajn te aplikacije, videćemo kasnije.

1. Pokrenite Microsoft Visual Studio .NET
2. Na stranici Start Page izaberite New Project (ili, na glavnom meniju, File -> New -> Project...)
3. Na New Project dijalogu u Project Types listi birajte **Visual C# Projects**
4. U Templates listi birajte **Windows Application (.Net)**

5. U Name tekst polju zamenite <Enter name> sa WinForms2
6. U kombo polju Location prihvatite predlog ili otkucajte sami

OK

7. Pritisnite F5 da bi testirali program
8. Zatvorite ga, i vraćate se u VS.

Na prikazu klasa primećujete da je kreirana klasa Form1 izvedena iz klase Form koja pripada prostoru imena System.Windows.Forms

```
public class Form1 : System.Windows.Forms.Form
```

Na ovaj način klasa Form1 nasleđuje (poseduje) sve osobine roditeljske klase. Ali, osim što sadrži sve osobine Form klase, možete joj dodeliti posebne metode, polja, koje su samo vama od interesa. Takođe, ovoj klasi možete dodavati i druge Widnows kontrole i time učiniti formu da izgleda i funkcioniše onako kako želite.

Naslovna linija - *Title bar*

Forma se sastoji iz različitih grafičkih celina koje zajedno čine da se cela forma ponaša tipično kao jedan prozorski objekat i kao objekat koji čuva ostale objekte tj. kontrole (kontejner kontrola). Gornji deo forme obično je označen posebnom bojom. Naziva se **naslovna linija – *title bar*** i čuva čitljivi naziv forme.

Ovaj tekst pripada svojstvu **Text** forme. Dakle, dodajete ga popunjavanjem ovog svojstva u prozoru *Properties*.

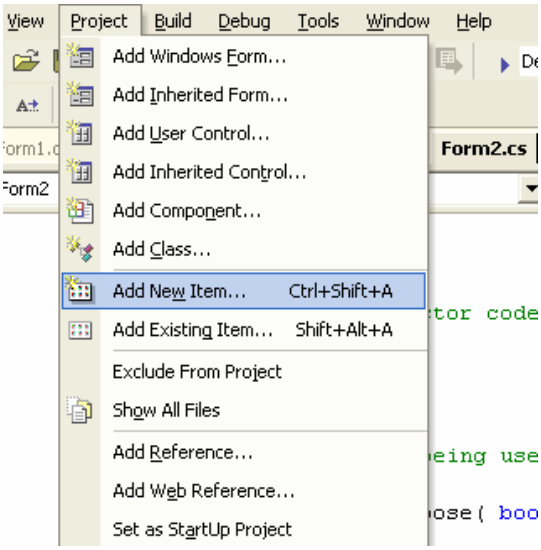
Ikona - *Icon*

Na levom delu naslovne linije forma prikazuje malu sliku koju nazivamo ikonom ili sistemskom ikonom. Visual Studio obezbeđuje podrazumevanu ikonu za sve forme. Ako pak želite da koristite drugu ikonu to možete izvesti biranjem *Icon* polja u prozoru *Properties* kada je forma selektovana, a zatim kliknite na dugme  . Tada se otvara *Open* dijalog pomoću koga možete naći novu ikonu koju će aplikacija nadalje koristiti. Inače, ako ikonu želite da promenite programski morate prvo ikonu programski kreirati. Ikona je u prostoru imena **System.Drawing**. Primer:

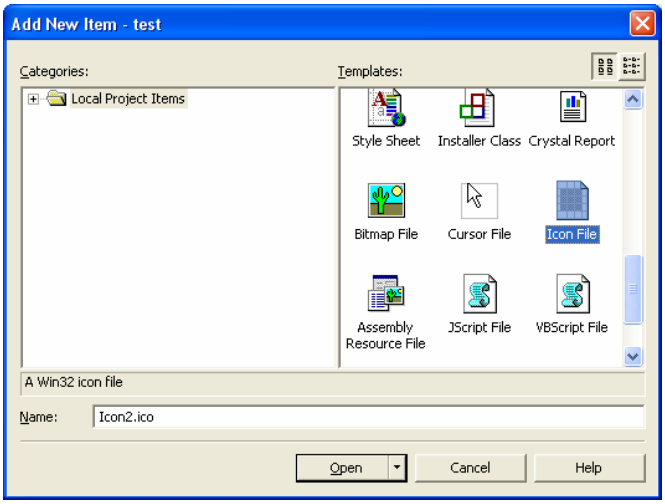
```
private void Form1_Load(object sender, System.EventArgs e)
{
   System.Drawing.Icon IC = new
System.Drawing.Icon("C:\\Programs\\alarm1.ico");
}
```

Ako dodajete novu ikonu vašoj aplikaciji uradite sledeće:

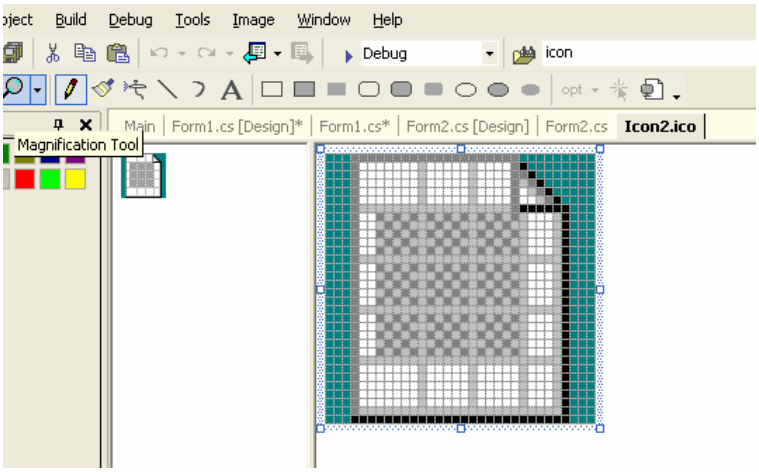
1. Na glavnom meniju odaberite *Project -> Add New Item...*



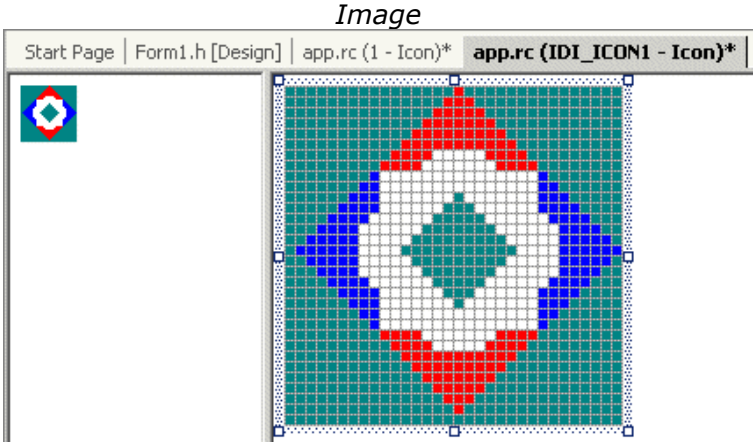
2. Kliknite na *Icon* a zatim kliknite na *Open*



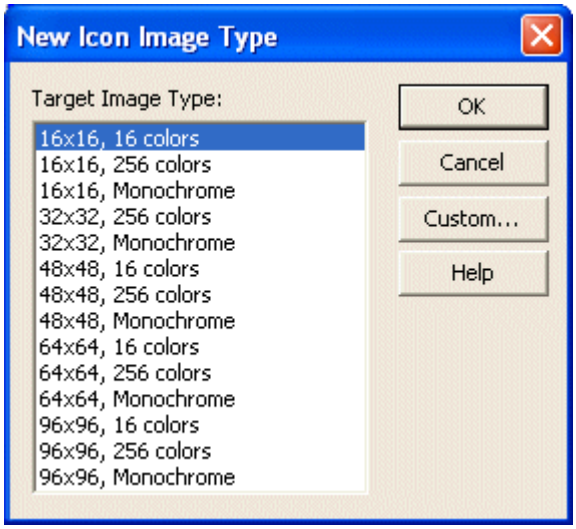
3. Otvara se editor za crtanje ikona, kao na slici



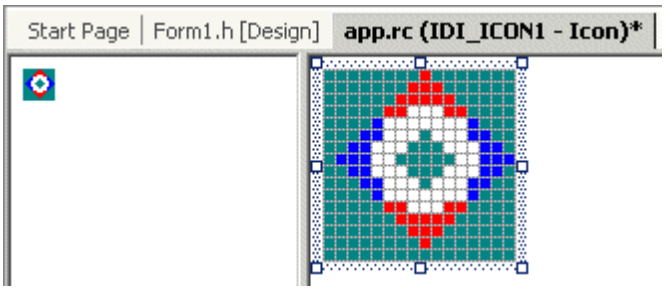
4. Nacrtajte ikonu koristeći boje sa palete Colors i alatke sa menija *Image*



5. Nastavljate sa dizajnom kontrole i izaberite na glavnom meniju Image -> New Image Type... Na dijalogu New Icon Image Type selektujte prvi izbor 16x16, 16 colors



6. Klik OK.
7. Da biste napravili ikonu transparentnom ostavite istu boju koja je bila na startu kao pozadinsku.



8. Ctrl + F5 i testirajte program

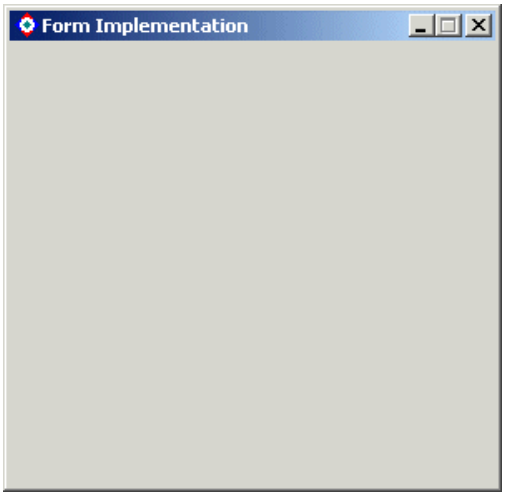
Sistemska dugmad

U krajnjem desnom uglu naslovne linije nalaze se sistemski dugmići: minimize, maximize, restore ili dugme za zatvaranje prozora. Postojanje ovih dugmića je takođe opciono, tj. nema smisla da se sve forme minimizuju, ovo naročito važi za modalne dijaloge, o čemu ćemo govoriti kasnije. Podešavanje ovih sistemskih dugmića izvodi se svojstvima **MinimizeBox** odnosno **MaximizeBox.**

Kako biste to uradili programski možete videti na primeru

```
private void Form1_Load(object sender, System.EventArgs e)
{
        System.Drawing.Icon IC = new
        System.Drawing.Icon("C:\\Programs\\alarm1.ico");
        Icon = IC;

        MinimizeBox = true;
        MaximizeBox = false;
}
```



Ako isključite dugmiće *minimize,* odnosno *maximize* to najverovatnije znači da ne želite da se dimenzije ovog dijaloga menjaju. Međutim to nažalost ne znači i to. I dalje to možete izvesti na isti način kao i ranije. Ovo svojstvo pripada delu fome koji se naziva **Borders**. Koristeći prozor Properties i svojstvo **FormBorderStyle** možete se igrati ivicama forme. Ako izaberete **FixedDialog** dobićete formu bez mogućnosti da menja svoje dimenzije

Na primer

```
private void Form1_Load(object sender, System.EventArgs e)
{
        System.Drawing.Icon IC = new
System.Drawing.Icon("C:\\Programs\\alarm1.ico");
        Icon = IC;
```

```
        MinimizeBox = true;
        MaximizeBox = false;

        FormBorderStyle = FormBorderStyle.FixedDialog;
}
```

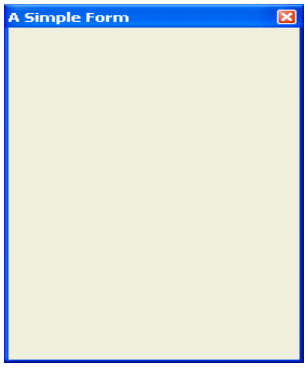
Ako želite od forme da napravite neku paletu alatki, tj. nekakav "tool", preporučuje se (poštujući nepisane "windows grafičke standarde") da forma ima samo malo *Close* dugme. *Tool* može biti sa ivicama koje dozvoljavaju promenu veličine, ili ne tj.

```
FormBorderStyle = FormBorderStyle.FixedToolWindow;
```

odnosno

```
FormBorderStyle = FormBorderStyle::SizableToolWindow;
```

Tool prozor ima naslovnu liniju male visine i nema ikonu, kao na slici



Klijentska oblast

Najveći deo forme pripada oblasti koju obično nazivamo **klijentskom** ili samo telo forme. Na tu površinu postavljamo kontrole ili je koristimo kao grafičku oblast preko koje ispisujemo tekst ili iscrtavamo nešto. Ova oblast mora imati inicijalno postavljeno svojstvo pozadinske boje ***BackColor***. Boju biramo iz palete ukoliko koristimo *Properties* prozor ili iz *Color* strukture programski:

BackColor = Color.AliceBlue;

BackgroundImage

Ako ipak više volite da u pozadinu postavite neku sliku onda koristite svojstvo ***BackgroundImage*** na primer

```
Bitmap Bg = new Bitmap("C:\\Programs\\test.bmp");
this.BackgroundImage = Bg;
```

WindowState

Kada je forma u veličini istoj kao kad je i postavljena u toku dizajna, kažemo da je stanje forme *normal.* Može se posebno podesiti da bude minimizovana ili maksimizovana kada se aplikacija pokrene. Ovo se podešava svojstvom ***WindowState***. Podrazumevana vrednost ovog svojstva je ***Normal***, tj. forma će se pojaviti na način na koji je podešena u dizajnu. Ukoliko pak programski želite da promenite izgled forme tj. da je minimizujete ili maksimizujete u nekom delu koda onda se to izvodi svojstvima ***Minimized*** ili ***Maximized***.

```
Fm.WindowState = FormWindowState.Maximized;
```

Ovo svojstvo se koristi i kod čitanja stanja forme. Drugim rečima, kada želite da proverite stanje vaše forme programski potrebno je da vrednost ovog svojstva poredite sa odgovarajućim vrednostima *Normal*, *Maximized* ili *Minimized*.

ShowInTaskbar

Kada kreirate jednu aplikaciju koja se sastoji od više formi (što je u praksi najčešći slučaj) morate da podesite da se sve forme ne vide u liniji na dnu ekrana – *task bar,* koja pokazuje koje su sve aplikacije otvorene (*taskbar presence*). Uobičajeno je da samo jedna (početna ili glavna) forma bude prikazana, dok se ostale moraju sakriti. Ovo se reguliše pomoću svojstva ***ShowInTaskbar*.**

Kreiranje objekta forme

Kreiranje forme obavljate na isti način kao i svih ostalih objekata. Klasa *Form* je u okviru prostora imena *Forms,* u okviru prostora imena *System*. Prikaz forme obezbeđujete metodom *Show.* Na primer

```
Form3 f = new Form3();
f.Show();
```

Zatvaranje forme

Kao što kreirate jednu formu da biste preko nje izveli neke UI akcije, tako je morate i zatvoriti pošto završite sa tim. Zatvaranje forme obavljate pomoću metode Close(). Sintaksa je

void Close();

Ako ovom metodom pokušate da zatvorite glavnu formu vaše aplikacije zatvarate i vašu aplikaciju

Pokretanje forme

Kada su dve ili više formi pokrenute na vašem računaru samo jedna može primati ulaz od korisnika (ovo smo već spominjali). Takva forma ima naslovnu liniju drugačije obojenu od ostalih i predstavlja aktivni prozor. Ostali prozori su neaktivni.

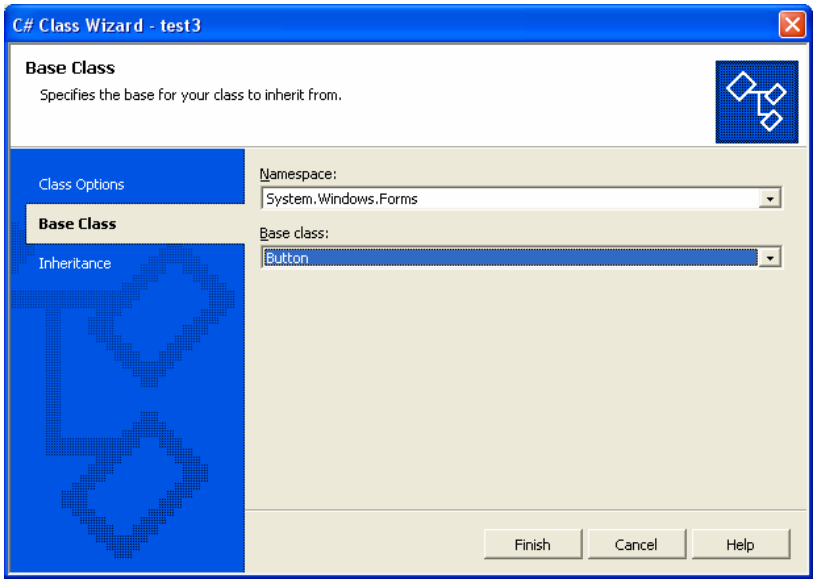
Događaj ***Activated***() detektuje kada neki prozor postaje aktivan. Pokretanje prozora izvodi se ili korisnički, putem miša na primer, ili programski metodom ***Activate().***

Nasuprot ovog događaja postoji i događaj koji registruje trenutak kada neka forma gubi korisnički ulaz. Ovo je događaj ***Deactivated***().

Primer: Napravite sopstvenu klasu koja će vam omogućiti da postavljate kružna dugmad u vašoj aplikaciji.

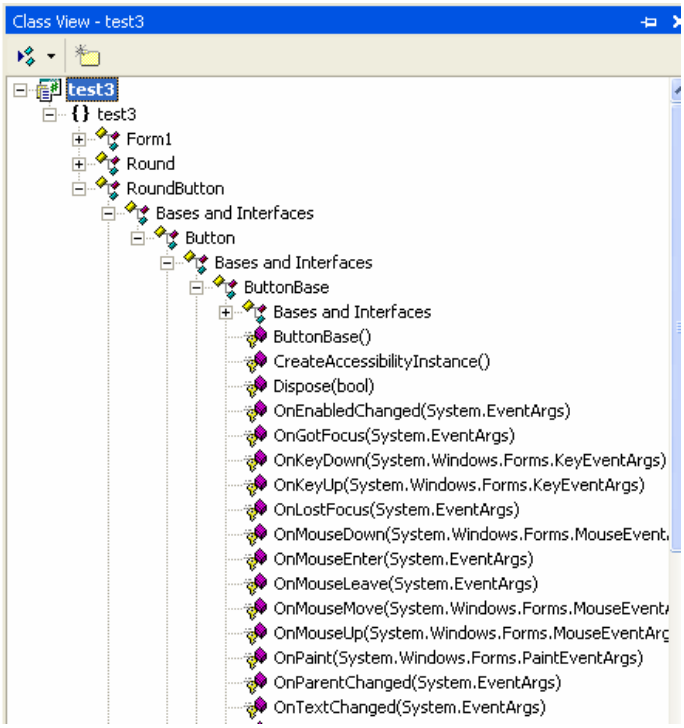
1. Treba da znate da je klasa koja je zadužena za kontrolu dugme u C# *Button* i pripada prostoru imena *System.Windows.Forms.* Ukoliko ovo niste ranije znali morate prvo naći informaciju o kontroli dugme iz Helpa koji vam stoji na raspolaganju dok radite sa IDE. Drugi način je da koristeći paletu sa kontrolama postavite dugme na formu, a zatim u kodu pronađete o kojoj klasi je reč.

2. Dodajte novu klasu koristeći IDE okruženje. Na ovom mestu obratite pažnju na karticu koja se tiče bazne klase. Pogledajte sliku

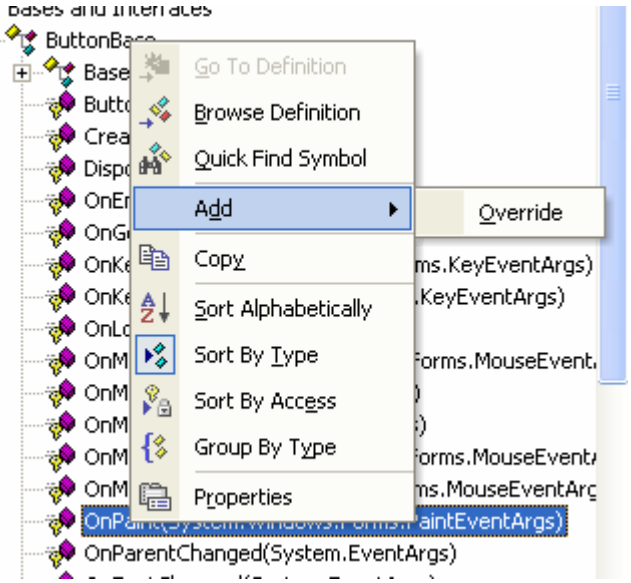


3. Pošto dugme mora da bude kružno mora se promeniti funkcija koja je zadužena za njegovo iscrtavanje i to tako da je van kruga površina transparentna a u krugu definisane boje.

Ovo znači da se virtuelni metod, koji je zadužen za iscrtavanje u klasi *Button,* mora promeniti tj. izvršiti nadjačavanje (redefinisanje) ovog metoda. Ovo se izvodi iz alatke *Class View* tako što otvorite *Bases and Interfaces* u roditeljskoj klasi *Button*, a zatim u sledećoj roditeljskoj klasi *ButtonBase*, gde nalazite metod OnPaint(...), kao na slici



Zatim, pritiskom na desni taster miša na metodi *OnPaint* birate stavku *Add,* a potom jedinu stavku *Override*, kao što je prikazano na slici:



Na ovaj način je dobijena funkcija zadužena za iscrtavanje dugmeta

```
protected override void OnPaint (System.Windows.Forms.PaintEventArgs
pevent)
{
}
```

Oblast koja se iscrtava definisana je posebnim grafičkim objektom. Njega samo pominjemo, bez dublje analize detalja. Detaljnije grafiku obrađujemo u posebnom poglavlju. Ostavljamo čitaocu da proba sam da dodaje nove osobine ovoj klasi.

Ukoliko samo ubacite promenu ovog parametra na mestu gde se iscrtava dugme, na sledeći način

```
    protected override void OnPaint(System.Windows.Forms.PaintEventArgs
pevent)
    {
        System.Drawing.Drawing2D.GraphicsPath el= new
        System.Drawing.Drawing2D.GraphicsPath();
        el.AddEllipse( 0,0,this.Width,this.Height );
        this.Region = new System.Drawing.Region( el );
    }
```

dobijate elipsasto dugme sa svim osobinama kao i kod pravougaonog. Jedino što ovo dugme nećete naći u paleti kontrola već ga morate ručno postavljati u kodu.

4. Postavite ovo dugme na formu na isti način kao i standardno pravougaono dugme. Prvo deklarišete sam objekat.

```
private test3.RoundButton rb;
```

Zatim dodate kôd za kreiranje kontrole i inicijalizaciju parametara koji se tiču pozicije, veličine kontrole. U primer je uključen i događaj *klik-miša*.

```
this.rb = new test3.RoundButton();
this.rb.Location = new System.Drawing.Point(10, 10);
this.rb.Name = "rb";
this.rb.Size = new System.Drawing.Size(262, 94);
this.rb.TabIndex = 1;
this.rb.Click += new System.EventHandler(this.rb_Click);
```

Ne zaboravite da dodate kontrolu kontejneru kontrola koji će pripadati formi, dakle

```
this.Controls.AddRange(new System.Windows.Forms.Control[] {
                                                    this.button1,
                                                    this.rb});
```

6. WINDOWS KONTROLE

Dugme

Dugme je Windows kontrola koja se koristi za iniciranje jedne akcije. Akcija se izvodi kada se mišem klikne na dugme. Često se klasično dugme naziva i komandno dugme. Naravno, postoje i druge kontrole koje mogu inicirati akciju tako što se klikne na njih baš kao i kod dugmeta. Vizuelno, dugme je kontrola od koje korisnik programa očekuje neku akciju kada izvrši „pritisak" na njega.

Sa stanovišta programera jedno dugme zahteva kontejner kontrolu. Kontejner može biti forma, toolbar ili nešto drugo.

Najčešće dugme u Windows aplikacijama je pravougaona kontrola koja prikazuje reč ili kratku rečenicu. Tekst na dugmetu obično opisuje akciju koja sledi pritiskom na to dugme. U .Net aplikacijama ova kontrola se realizuje korišćenjem ***Button*** kontrole. Postavlja se na klasičan način korišćenjem liste *toolbox* ili programski. Kada postavite neko dugme na formu, nadalje možete izvesti njeno podešavanje korišćenjem svojstava ove kontrole preko *Properties* prozora.

Text

Najbitnije svojstvo dugmeta za korisnika je poruka na njemu. Ova poruka zapisana je u svojstvu koje se naziva **Text**. Tekst se uvek prikazuje na vrhu kontrole i može biti dodatno formatiran. Uobičajeno je da opisuje akciju koju dugme pokreće/izvršava.

Najčešći tekstovi koje dugmad prikazuju su OK i Cancel. Poruka OK se obično prikazuje na dijalozima koji informišu korisnika o nekoj grešci, nekom stanju, ili je to samo informacija koja zahteva potvrdu od korisnika. Ukoliko se korisniku daje mogućnost da odustane od nastavka neke akcije obično se nudi Cancel dugme.

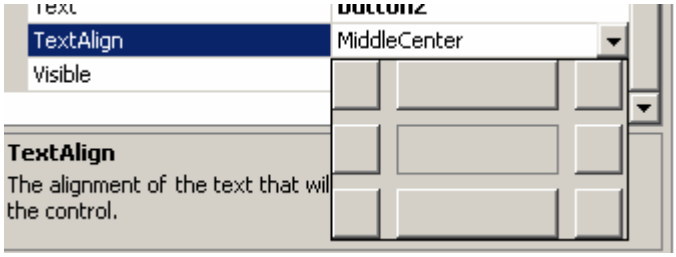
Naravno, tekst na dugmetu može se programski menjati u toku izvršavanja aplikacije, u zavisnosti od stanja u kome se aplikacija nalazi, na primer:

```
Button1.Text = "Let Me Go!";
```

Postoji još jedno svojstvo veoma značajno upravo za dugme. To je *podrazumevano dugme*. Samo jedno dugme na kontejneru kontrola može biti podrazumevano. Ono ima to svojstvo da pritiskom na taster *Enter*, kada je aktivna forma tog dugmeta, biva pritisnuto, tj. startuje se akcija koju ono inicira. Tačnije, ako neko drugo dugme ima fokus (vidljiv je tačkasti pravougaonik na njemu) izvršiće se akcija tog dugmeta. Inače, izvršava se akcija dugmeta koje je "podrazumevano" (vidljiva je tamna ivica).

TextAlign

Mesto teksta na dugmetu određuje svojstvo **TextAlign**, kao na slici ispod.



DialogResult

Kada korisnik klikne na neko dugme da bi zatvorio dijalog važno je da u kodu postoji informacija o tome na koje dugme je kliknuto. .Net okruženje pruža mehanizam kojim se može izvesti ova identifikacija. Ovo se postiže korišćenjem svojstva **DialogResult**. Vrednost može biti neka iz određenog skupa koji je definisan. Moguće vrednosti su *None, OK, Cancel, Abort, Retry, Ignore, Yes,* i *No*. Podešavanjem neke od ovih vrednosti preko *Properties* prozora definišete vrednost koju dijalog vraća, ako se pritiskom na vaše dugme izvede zatvaranje istog dijaloga.

Image, ImageAlign, ImageList, ImageIndex

U grupu dugmadi pripadaju i **bitmapirana dugmad**. Ove kontrole imaju osobine dugmadi ali prikazuju i sliku tj. bitmapu, a mogu prikazati i naslov na vrhu. Da biste napravili ovakvo dugme morate imati na raspolaganju neku bitmapu, ili je morate sami kreirati. Zatim, u prozoru Properties, koristeći polje **Image**, birate željenu bitmapu ili ikonu. Kada dodate sliku treba da postavite sliku na željeni način koristeći **ImageAlign** svojstvo. Drugi način je da se koristi **ImageList**

svojstvo koje se kreira na osnovu nekoliko slika, a zatim se koristeći **ImageIndex** specificira koja slika će biti prikazana.

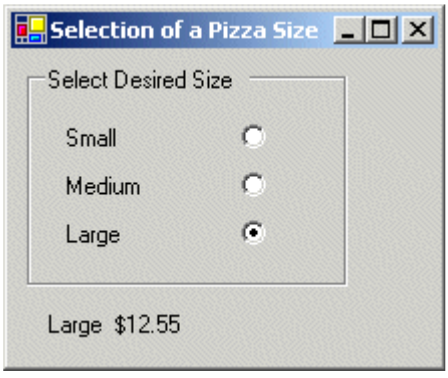
FlatStyle

Standardno, dugme se prikazuje kao da je izdignuto. Ako želite da promenite ovakav izgled menjaćete svojstvo **FlatStyle:**

- Flat: Dugme je prikazano kao ravno. Kada se mišem pređe preko njega ono postaje istaknuto (*highlighted*);
- Popup: Dugme se prikazuje kao ravno. Ipak, kada se mišem stane na njega ivice dugmeta se pojavljuju tako da dugme tada izgleda izdignuto;
- Standard: Standardni izgled na koji ste već navikli i standardno ponašanje;
- System: Zavisi od operativnog sistema koji se koristi – za detalje obratiti se MSDN-u.

Radio-dugme

Radio-dugme, neko ga naziva još i opciono dugme, je kontrola kružnog oblika koja se pojavljuje u grupama sa drugima kontrolama istog tipa, tj. više radio-dugmadi pojavljuju se zajedno. Svako dugme sastoji se od malog praznog kruga i labele koja opisuje dugme, tj. daje korisniku neku informaciju o akciji. Kada korisnik pritisne jedno od dugmića koji su u grupi ono dobija oblik ⊙ a ostali ostaju prazni. Ovo znači da se grupom radio-dugmića realizuje korisnička akcija izbora samo jedne stavke iz grupe.



Kontrola je bazirana na klasi **RadioButton**. Kreiranje kontrole izvodite programski, koristeći konstruktor klase i postupke koje smo već pominjali, ili u toku dizajna jedne forme (ili nekog drugog kontejnera kontrola) izborom sa paleta Toolbox. Radio-dugme može da postoji samo u grupi, zato je potrebno uključiti svu radio-dugmad u neku drugu kontrolu koja i vizuelno može odvojiti grupu radio-dugmića od drugih kontrola, odnosno drugih grupa radio-dugmadi. Najčešća kontrola koja se koristi za to je ***GroupBox*** kontrola.

Checked

Ovo svojstvo definiše da li je radio-dugme selektovano ili ne. Ukoliko jeste, to znači da su sva ostala neselektovana. Osim za čitanje stanja u kome se dugme nalazi isto svojstvo možete upotrebiti i u "suprotnom pravcu". Ovim svojstvom možete programski podešavati izbor nekog radio-dugmeta. Na primer:

```
private void Form1_Load(object sender, System.EventArgs e)
{
        this.radioButton2.Checked = true;
}
```

CheckAlign

Podrazumevano podešavanje radio-dugmeta je tako izvedeno da se kružić nalazi sa leve strane od teksta koji je pridružen ovoj kontroli. Kao i kod polja za potvrdu, i ovde postoji puno mogućnosti za fino podešavanje prikaza. Pozicija kružića u odnosu na tekst vrši se pomoću svojstva **CheckAlign**. Moguće vrednosti su: **TopLeft**, **TopCenter**, **TopRight**, **MiddleRight**, **BottomRight**, **BottomCenter**, odnosno **BottomLeft**. Programski podešavanja možete izvesti kao na primeru:

```
private void Form1_Load(object sender, System.EventArgs e)
{
    this.radioButton1.CheckAlign = ContentAlignment.MiddleRight;
}
```

Appearance

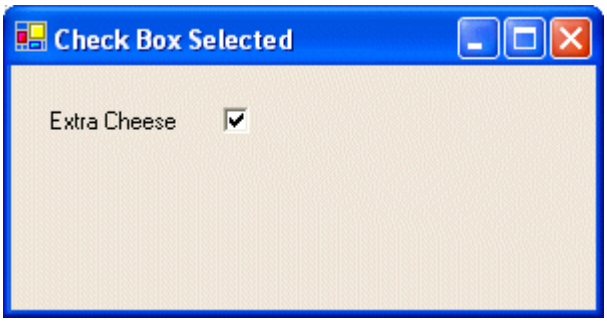
Podrazumevano podešavanje izgleda radio-dugmeta je takvo da je ono ispunjeno velikom tačkom ako korisnik selektuje to dugme. Opciono, možete podesiti da ono radi kao "toggle" dugme. Šta to znači? Samo jedno dugme u grupi može biti u donjem položaju (stanje selektovano) dok su ostala u gornjem položaju. Ako se neko drugo dugme pritisne ono ide dole, a dugme koje je prethodno bilo u donjem položaju ide gore, tj. sva ostala su otpuštena.

Promena podrazumevanog prikazivanja ove kontrole izvodi se preko svojstva **Appearance**. Programski to možeti izvesti kao na primeru:

```
private void Form1_Load(object sender, System.EventArgs e)
{
        this.radioButton1.Appearance = Appearance.Button;
}
```

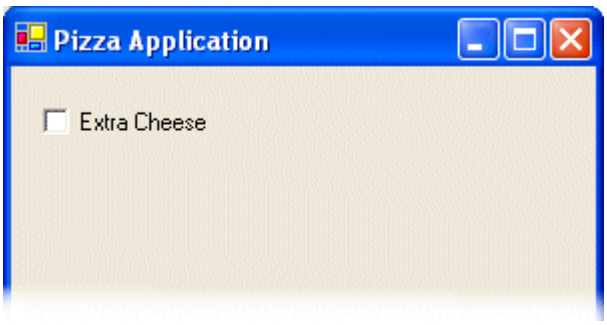
Polje za potvrdu

Polje za potvrdu je kontrola koja daje samo dve opcije: potvrđeno ili prazno. Vizuelno ova kontrola prikazuje malo kvadratno polje na koje korisnik može da klikne (tačnije, korisnik može da klikne na celu kontrolu – uključujući tekst). Na startu polje je prazno ☐. Ako korisnik klikne na njega oznaka o potvrdi pojavljuje se u kvadratnom polju ☒. Da bi se korisniku omogućilo da zna šta polje za potvrdu predstavlja kontroli je pridružena labela koja prikazuje tekst. Kada je kvadratno polje prazno ☐, vrednost je **false**. Kada je polje ispunjeno oznakom za potvrdu ☒, vrednost je **true**.



Ovaj objekat definisan je klasom CheckBox.

1. Kreirajte novu Visual C# Windows Application.
2. Promenite inicijalni naziv forme.
3. Da dodate polje za potvrdu kliknite na CheckBox dugme unutar liste *toolbox*, a zatim na formu.
4. Promenite svojstvo *Name*. Ovo svojstvo se menja preko *Properties* prozora. Ukucajte **cbExtraCheese** i pritisnite *Enter.*
5. Kliknite na *Location*. Unesite **16, 16** i pritisnite *Ente.r*
6. Kliknite na *Text*. Unesite **&Extra Cheese.**
7. Pokrenite aplikaciju.



Checked

Podrazumevano ponašanje ove kontrole je da se pojavljuje neštriklirana (nepotvrđena), tj. njena vrednost je inicijalno postavljena na *False*. Da bi se promenila korisnik treba da klikne na kontrolu. Međutim, ako želimo da je svojstvo inicijalno postavljeno na vrednost *True,* tj. da kontrola pri prvom pojavljivanju bude potvrđena, možemo ili promeniti vrednost svojstva **Checked** na **true**, ili programski to učiniti na sledeći način:

```
private void Form1_Load(object sender, System.EventArgs e)
{
        this.checkBox1.Checked = true;
}
```

Da biste proverili da li je polje selektovano treba da pročitate vrednost svojstva **Checked**.

Alignment

Podrazumevano, kvadratić se pojavljuje na levoj strani labele. .Net okruženje vam pruža puno različitih izbora kojima možete podešavati prikaz kontrola. To se na ovoj kontroli može potvrditi.

Najčešća podešavanja tiču se pozicije kvadratića u odnosu na tekst cele kontrole. Ta podešavanja se izvode pomoću svojstva **CheckAlign**. Moguće vrednosti su: **TopLeft**, **TopCenter**, **TopRight**, **MiddleRight**, **BottomRight**, **BottomCenter**, i **BottomLeft**.

Programski podešavanja izvodite postavljanjem vrednosti **CheckAlign**, ali preko enumeratora **ContentAlignment**, na primer:

```
private void Form1_Load(object sender, System.EventArgs e)
{
        this.checkBox1.CheckAlign = ContentAlignment.MiddleRight;
}
```

CheckState

Neretko, kontrola Checkbox osim stanja *True,* odnosno *False*, treba da pokaže i neko međustanje. Ovo stanje obično se naziva "half-checked" ili u nekom slobodnom prevodu polupotvrđeno. U tom stanju polje za potvrdu izgleda kao da je onemogućeno (*disabled*). Ovakvo specifično ponašanje kontroliše se preko svojstva **CheckState**. Da biste podesili međustanje postavite vrednost ovog svojstva na **Indeterminate** ili programski:

this.checkBox1.CheckState = CheckState.Indeterminate;

ThreeState

Jedino svojstvo **CheckState** dozvoljava postavljanje polja za potvrdu u međustanju. Ako želite da se koriste tri stanja dugmeta za potvrdu, na primer označeno, poluoznačeno i neoznačeno, koristite svojstvo **ThreeState**.

Postavljanjem svojstva **CheckState** (Boolean tipa) na vrednost *True* korisnik treba da klikne dva do tri puta do željene vrednosti. Dobijanje stanja dugmeta za potvrdu u ovom slučaju izvodite preko svojstva **Indeterminate**.

Primer postavljanja nekog polja za potvrdu tako da ima tri stanja:

```
private void Form1_Load(object sender, System.EventArgs e)
{
        this.checkBox1.Checked = true;
        this.checkBox1.ThreeState = true;
```

```
}
```

Appearance

Očekivani izgled ove kontrole je kvadratić koji može da primi komandu od korisnika aplikacije ili prikaže neku tekuću vrednost. Međutim, alternativa je da polje za potvrdu prikažete i kao standardno dugme. U tom slučaju kada kliknete na njega ono ostaje u donjem položaju, tj. položaj dugmeta dole ili gore određuje u stvari stanje polja za potvrdu *True* odnosno *False*. Ako korisnik klikne na dugme još jednom ono se podiže u gornji položaj.

Da biste promenili standardni prikaz treba promeniti svojstvo **Appearance**. Sa vrednosti **Normal** prebaciti na vrednost **Button**. Programski to možete učiniti ovako:

```
private void Form1_Load(object sender, System.EventArgs e)
{
        this.checkBox1.Appearance = Appearance.Button;
}
```

TEKST kontrole

Labela

Labela je kontrola za prikaz teksta. Korisnik ne može direktno da menja tekst u ovoj kontroli. Labela može biti korišćena sama na formi, za ispisivanje tekstualnih poruka u toku izvršavanja programa, ili stajati uz neku drugu kontrolu koju svojim tekstom dodatno opisuje.

Da biste kreirali jednu labelu kliknite na dugme *Label* na listu *Toolbox* a zatim na formu.

Programski, kreiranje ove kontrole ide na način koji smo već upoznali. Koristeći operator **new** kreirate objekat koristeći podrazumevani konstruktor. Kada kreirate ovu kontrolu podesite njena svojstva, na primer:

```
private void Form1_Load(object sender, System.EventArgs e)
{
        Label lblFirstName = new Label();
```

```
            lblFirstName.Text = "&First Name";
            lblFirstName.Location = new System.Drawing.Point(20, 80);
            this.Controls.Add(lblFirstName);
        }
```

TextBox

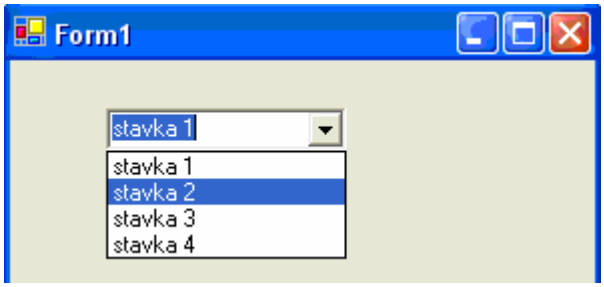
Kontrola tekst polje koristi se za prikaz tekstualnih poruka ili za unošenje teksta. Tekst polje je kontrola pravougaonog oblika. Površina za prikaz, odnosno unos teksta, zauzima celu površinu kontrole. Zbog toga uz ovo tekst polje, uobičajeno, ide labela kojom se opisuje namena tekstualnog polja.

Kreiranje ove kontrole izvodi se tako što kliknete na **TextBox**, a zatim na formu.

Tekst polje kontrola bazirana je na klasi TextBox. Uz pomoć ove klase može se programski kreirati i modifikovati novo tekstualno polje.

Kombo polje

Kombo polje je standardna grafička kontrola koju ste sigurno već sreli. Kontrola sadrži listu stringova koje korisnik aplikacije može videti pre nego što izabere samo jednu sa liste. Tipično, kombo polje je kontrola pravougaonog oblika sa tekstualnim poljem i jednim dugmetom sa strelicom nadole. Ovim dugmetom otvara se lista stavki koje stoje na raspolaganju. Kada otvori listu korisnik jednostavnim klikom na željenu stavku, ili koristeći tastere strelice nadole i gore (pritiskom na Enter), bira željenu stavku iz liste.



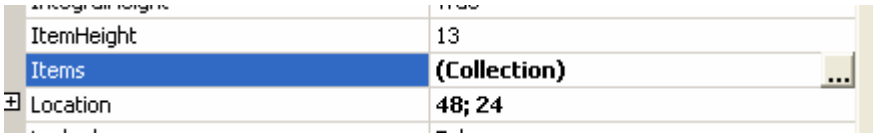
Stavka koju je korisnik izabrao pojavljuje se u tekst polju kombo polja.

Postoje različita podešavanja kojima se može podesiti izgled i ponašanje ove kontrole. Pretraga uz pomoć početnih slova takođe je uključena u ovu kontrolu i veoma je značajna ako je broj stavki u listi veliki, na primer ako je kontrola vezana za neki bazu podataka.

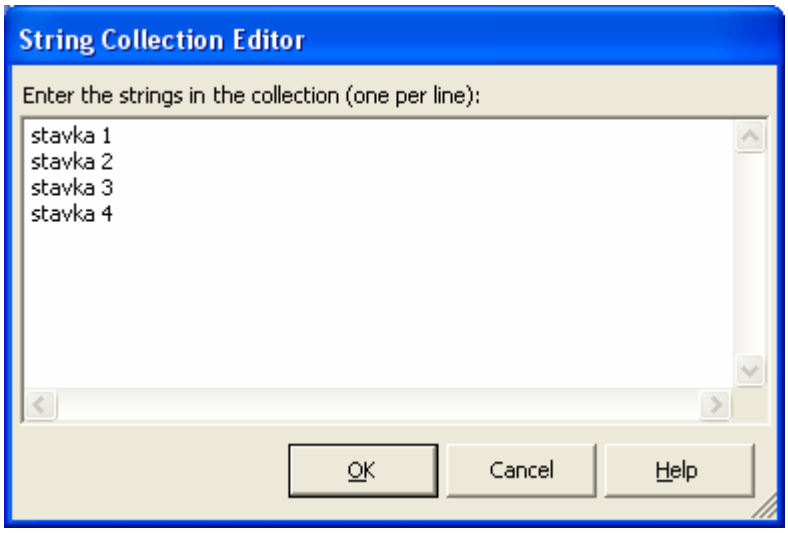
Standardno, kreiranje kontrole izvodite preko Toolbox palete ili programski. Dodavanje stavki u listu najčešće se obavlja samo programski. Na primer:

```
private void Form1_Load(object sender, System.EventArgs e)
{
        this.comboBox1.Items.Add("Small");
        this.comboBox1.Items.Add("Medium");
        this.comboBox1.Items.Add("Large");
        this.comboBox1.Items.Add("Jumbo");
}
```

Naravno da stavke kombo polja koje se neće menjati možete dodati i preko *Properties* prozora i polja *Collection*, kao na slici ispod.



Pritiskom na dugme ... otvara se dijalog za unos stavki liste, kao na slici:



Stavke se u listu ove kontrole unose kao pojedinačne linije teksta.

Pogledajmo kako se programski utvrđuje stavka koju je korisnik odabrao. Ovo je nezaobilazni deo kada se koristi ova kontrola. Za dobijanje vrednosti koju je korisnik izabrao koristi se svojstvo **Text**. Isto svojstvo može se koristiti i za prikaz izabrane vrednosti:

```
        private void comboBox1_SelectedIndexChanged(object sender,
System.EventArgs e)
        {
                this.label1.Text = this.comboBox1.Text;
        }
```

Kombo box koristi listu stavki i njihovih indeksa za izbor i prikaz. To znači da možete koristiti tekst ili indeks neke stavke da biste saznali šta je korisnik izabrao, ili da bi programski podesili vrednosti.