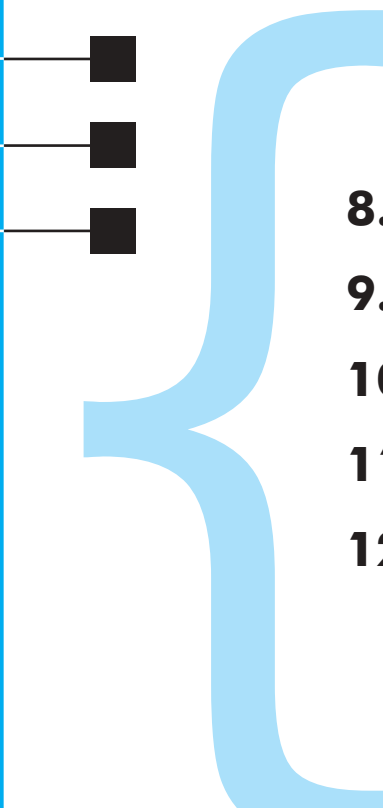




Dijelovi .NET-a



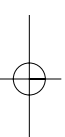
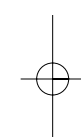
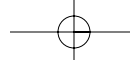
8. POGLAVLJE: WINDOWS FORMS

9. POGLAVLJE: ADO.NET

10. POGLAVLJE: ASP.NET

11. POGLAVLJE: XML

12. POGLAVLJE: WEB-SERVISI



8. POGLAVLJE

Windows Forms

U ovom poglavlju:

- Kada pisati prozorske aplikacije
- Kako napraviti aplikaciju za Windowse
- Kontrole u kolekciji klasa Windows Forms
- Manipulacija formama
- Crtanje i ispis na pisač
- Priprema aplikacija za distribuciju

Nekoć, u doba dok je Internet bio tek igračka zaigranih entuzijasta, razvojni alati u Windowsima bavili su se isključivo izradom prozorskih aplikacija. Pod tim pojmom podrazumijevamo sve aplikacije koje kao osnovu sučelja koriste prozore (po kojima su Windowsi i dobili ime) odnosno, kako se oni stručnije nazivaju, forme.

Glavna odlika prozorskih aplikacija je što se one nalaze i pokreću na lokalnom računalu. Takvih aplikacija na svom računalu sigurno imate pregršt – od jednostavnih Notepada i Calculatora, preko pomoćnih programa kao što su Internet Explorer, WinAmp ili Acrobat Reader do velikih aplikacija i paketa poput Microsoft Officea, Corel Drawa i AutoCAD-a.

Prozorske aplikacije najčešće dolaze na instalacijskim CD-ima ili ih, ukoliko se radi o manjim aplikacijama, možete skinuti s Interneta. U oba slučaja potrebno ih je instalirati ili

III. DIO: DIJELOVI .NET-A

raspakirati na računalu, a kasnije, kada ih zatrebate, samo ih pokrenete. Najčešće se radi o izvršavanju neke datoteke s nastavkom ".exe", što često puta radimo i ne znajući zahvaljujući prečicama (engl. *shortcut*) koje pokazuju na te datoteke. No sve ste to, vjerujemo, već znali...



Prozorske aplikacije u ovoj knjizi koristimo kao sinonim za aplikacije pisane za Windows. Takve aplikacije se u stručnoj literaturi nazivaju i *rich client* aplikacije.

Širenjem Interneta i pojavom sve većeg broja brzih i praktički stalnih veza postepeno se smanjuje potreba za prozorskim aplikacijama. Brojne funkcionalnosti nekoć rezervirane isključivo za prozorske aplikacije dostupne su u obliku web-stranica. Primjerice, svom sandučiću elektroničke pošte možete pristupiti koristeći prozorsku aplikaciju za *e-mail* ili preko često jednako funkcionalnog web-sučelja. Drugi zgodan primjer su intranetski sustavi u poduzećima, koji sve češće pružaju mogućnosti koje su prije bile domena prozorskih poslovnih aplikacija.



Prozorske aplikacije mogu biti samostojeće (engl. *stand-alone*) ili klijentsko-serverske (engl. *client/server*). Samostojeća aplikacija je ona kojoj su svi podaci i sva logika smješteni na istom računalu. Klijentsko-serverske aplikacije u svom radu zahtijevaju vezu s nekom drugom, serverskom aplikacijom, najčešće smještenom na nekom drugom računalu u mreži.

Naravno, to ne znači da se prozorskim aplikacijama crno piše. Postoji velik broj situacija u kojima one predstavljaju bolje ili čak jedino rješenje. Spomenimo samo računalne igre, multimedijalne svirače, uredske aplikacije, grafičke programe, razne klijente za internetske servise... U krajnjoj liniji, ako ništa drugo, uvijek će postojati preglednik web-stranica koji je i sam prozorska aplikacija.

U ovom ćemo poglavlju naglasak staviti na biblioteku klasa Windows Forms, koja predstavlja svojevrsnu nadogradnju sustava koji se dosad koristio. Naime, za izradu korisničkog sučelja u starijim se razvojnim alatima koristio znatno kompliciraniji Win32 API (kratica za *application program interface*). Osim Windows Formsa, pozabavit ćemo se i klasama u biblioteci GDI+ koja omogućava crtanje po ekranu. Krenimo redom...

Forme i kontrole

Vjerujemo da ste se u sučelju Windowsa toliko udomaćili da ni ne primjećujete njegove sastavnice. Prva stvar koju treba uočiti je forma. Forme su osnova korisničkog sučelja i možemo ih poistovjetiti sa prozorima.

Korisničko sučelje gradimo tako da na formu postavljamo kontrole. Kontrole su zajedničko ime za natpise, gumbе, kućice za upis, padajuće liste i razne druge oblike prezentacije informacija i mogućnosti. Velik broj kontrola predefiniран je na nivou operativnog sustava, no moguće je dodavati nove kontrole, modificirati postojeće te stvarati vlastite.

Peripetije u pozadini

Forma nije ništa drugo nego objekt stvoren instanciranjem određene klase. Sjetimo se – klasa je nacrt prema kojem se stvara (instancira) objekt. Klasa na temelju koje stvaramo standardnu formu je `System.Windows.Forms.Form`, što možemo vidjeti iz sljedećeg primjera:

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

Sad postaje kompliciranije. Svaka forma je ujedno i klasa. Drugim riječima, stvaranjem objekta forme stvaramo i novu klasu. Nova klasa nasljeđuje sve karakteristike osnovne klase (dakle, klase `System.Windows.Forms.Form`) i ako na njoj ne napravimo nikakvu promjenu, one će biti identičnih karakteristika. Međutim, promijenimo li našoj formi karakteristike (mijenjamo svojstva, dodajemo kontrole), mijenjat će se i karakteristike naše klase i ona će se razlikovati od osnovne koju smo inicijalno naslijedili.

Zamislamo sada da u programu imamo potrebu za novom formom. Imamo dvije mogućnosti – kreirati novu formu na temelju klase `System.Windows.Forms.Form` ili na temelju klase

naše izmijenjene forme. Napravimo li to na prvi način, dobit ćemo praznu formu (jer klasa `System.Windows.Forms.Form` sadrži nacrt za praznu formu). Ako pak naslijedimo našu klasu, nova će forma imati sve karakteristike koje ona sadrži – sva svojstva, sve kontrole i sve ostalo:

```
public class Form2 :
NasaAplikacija.Form1
```

A što je s klasama? Kada kreiramo instancu kontrole na formi, koristimo klasu koja je opisuje i kreiramo objekt, no ne i klasu:

```
this.label1 = new
System.Windows.Forms.Label();
this.button1 = new
System.Windows.Forms.Button();
```

Za svaku kontrolu postoji druga klasa, no svi-ma je zajedničko da nasljeđuju osnovnu klasu `Control`; ona sadrži najosnovnije karakteristike koje kontrola mora imati. Štoviše, i klasa `System.Windows.Forms.Form` inicijalno nasljeđuje klasu `Control`, pa možemo reći da je i forma – kontrola.

Kao što smo već vidjeli u četvrtom poglavlju, na izgled i ponašanje formi i kontrola utječemo mijenjanjem njihovih svojstava i povezivanjem funkcionalnosti na događaje koje stvaraju. Kako kontrola

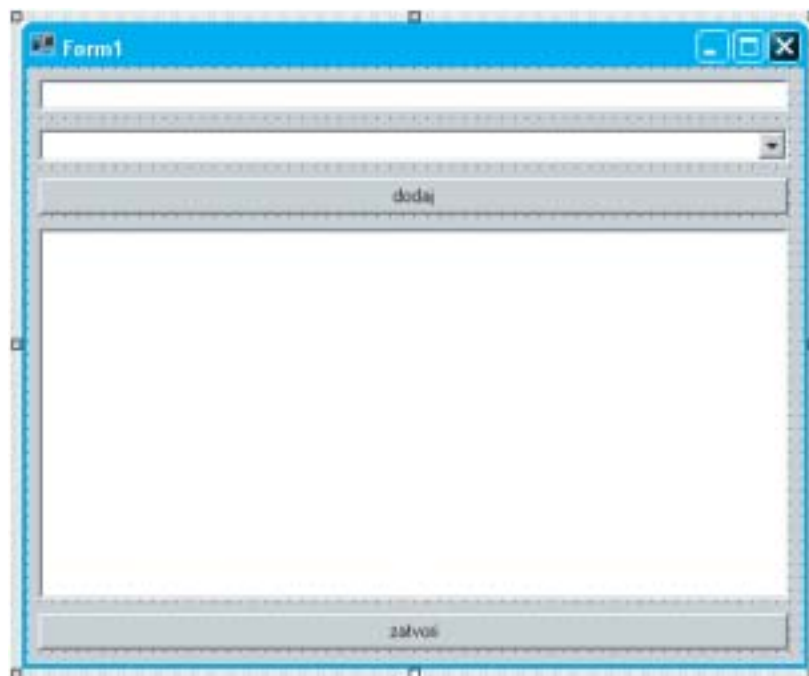
III. DIO: DIJELOVI .NET-A

ima puno, a svojstava, metoda i događaja još više, čak i površno šetanje kroz njih zauzelo bi prilično prostora. Zato smo se odlučili upoznati vas s kontrolama kroz nekoliko jednostavnih i složenijih primjera. Ne samo da ćete upoznati velik dio kontrola i njihova svojstva već ćemo zajedno proći kroz proces izrade aplikacija.

Primjer: Beskorisna aplikacija

Za početak ćemo napraviti beskorisnu aplikaciju za unos podataka. Kažemo “beskorisno” jer se podaci u njoj neće pamtit i pa će, jednom kada iz nje izađete, svi uneseni podaci biti izgubljeni. Ipak, naučit ćemo kako složiti jednostavno korisničko sučelje, koristiti gumbe, polja za upis, padajuće liste i još mnogo toga.

Slika 8-1:
Beskorisna aplikacija u dizajnerskom načinu

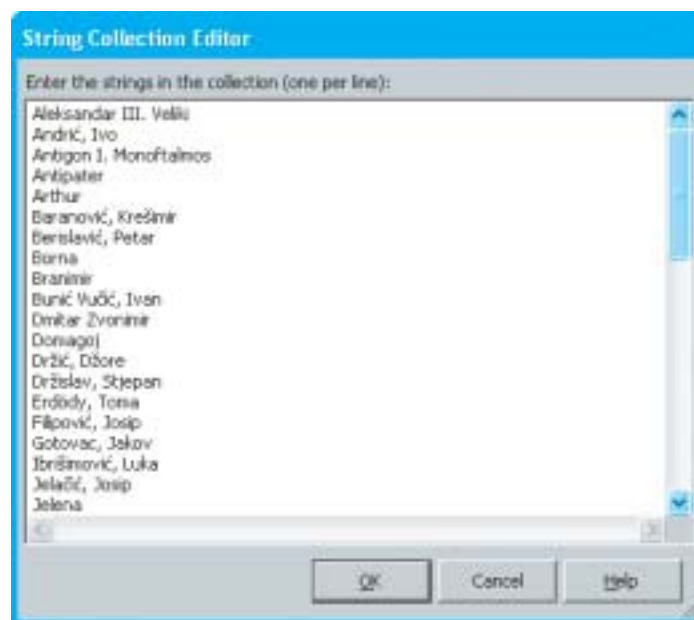


Čitanje knjiga u krevetu ili na plaži je super, no preporučujemo vam da dok prolazite kroz ovo i sljedeća poglavlja budete za računalom, imate otvoren Visual Studio .NET 2003 i isprobavate sve što spominjemo. Tako se puno bolje uči.

8. POGLAVLJE: WINDOWS FORMS

Kreirajmo novi projekt za prozorsku aplikaciju i na praznu formu dovucimo sljedeće kontrole: TextBox, ComboBox, ListBox i dvije kontrole tipa Button. Cilj nam je napraviti sljedeće: dozvoliti korisniku da upiše neki izraz u polje za upis, odabere neku vrijednost iz padajuće liste te klikom na gumb doda kombinaciju tih dvaju izraza na listu. Preostali gumb poslužiti će nam za demonstraciju zatvaranja prozora odnosno izlaska iz aplikacije.

Prvo moramo promijeniti nekoliko svojstava. Kontrolu textBox1 svojstvo Text postavite na praznu vrijednost (izbrišite postojeće), dok gumbima isto svojstvo postavite na vrijednosti “dodaj” za button1 i “zatvori” za button2. Oko padajuće liste ćemo imati nešto više posla – prvo ju je potrebno napuniti podacima. To ćemo napraviti tako da u redu svojstva Items kliknemo na gumbić s tri točkice, koji će se pokazati nakon što taj red označimo. Klikom na njega otvorit će se prozor s velikim poljem za upis. U njega upisujemo ponuđene vrijednosti – svaku vrijednost u svoj red, kao što možemo vidjeti na slici 8-2. Konačno, svojstvo DropDownStyle postavite na DropDownList.



Slika 8-2:
*Izbore u padajućem
izborniku upisujemo
svaki u poseban red.*

Sljedeći korak je dodavanje funkcionalnosti gumbima. Prvi gumb (button1) služiti će za prebacivanje unesenog i odabranog teksta u listu. Stoga trebamo dvokliknuti na njega i automatski će se kreirati funkcija unutar koje ćemo napisati kôd koji će napraviti ovo što smo upravo spomenuli. Istu smo stvar mogli napraviti i na drugi način – označiti gumb, u pomoćnom prozoru kliknuti na simbol munje

III. DIO: DIJELOVI .NET-A

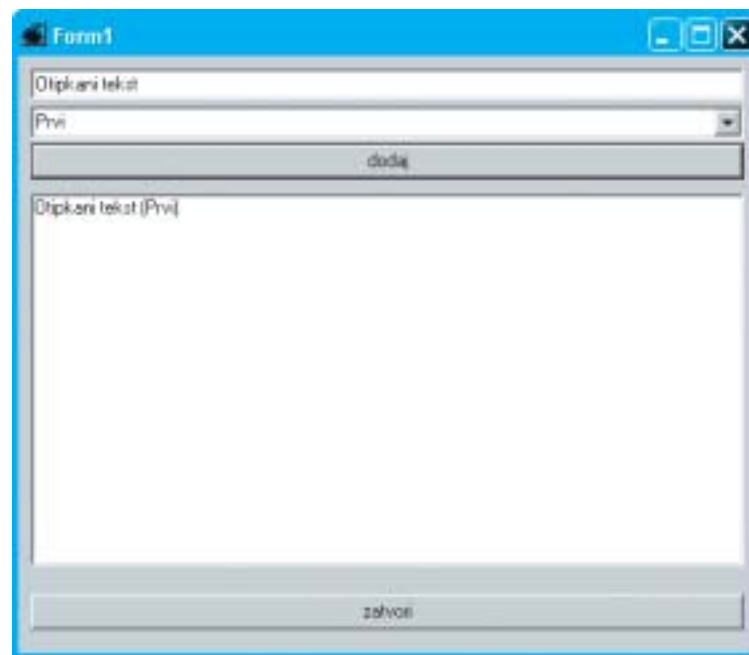
kako bismo ušli u događaje vezane uz taj gumb te dvokliknuti na bijeli dio reda u kojem piše Click. Međutim, kako je događaj Click glavni događaj za kontrolu tipa gumb, dovoljno je na njega dvokliknuti. (Isto vrijedi i za ostale kontrole, samo što one mogu imati neki drugi glavni događaj.)

Kôd koji ćemo dodati u tako dobivenu funkciju glasi ovako:

```
listBox1.Items.Add(textBox1.Text + " (" + comboBox1.Text + ")");
```

Što taj komad kôda radi? Kontrola tipa ListBox ima zadaću držati i prikazivati listu izraza. Ta lista izraza smještena je u kolekciju Items pa kada joj želimo pristupiti pišemo izraz listBox1.Items. Kolekcija Items sadrži metodu Add pomoću koje joj dodajemo novi zapis u kolekciju (odnosno novi izraz na listu). Sve što navedemo kao parametar te metode (u zagradi) bit će dodano na kraj liste.

Slika 8-3:
Beskorisna aplikacija
prilikom izvršavanja



Izrazom textBox1.Text pristupamo vrijednosti koja je upisana u polje za unos textBox1, dok izrazom comboBox1.Text pristupamo vrijednosti koja je odabrana u padajućoj listi. Spajanjem tih dvaju izraza i dodavanjem zagrada iz kozmetičkih razmaka kreiramo izraz koji ćemo dodati u listu. Drugim riječima, ukoliko je u polju za upis upisan izraz “Otipkani tekst”, a iz padajuće liste odabrana vrijednost “Prvi” onda će redak u listi nakon pritiska na gumb izgledati ovako: “Otipkani tekst (Prvi)”. Uostalom, pokrenite aplikaciju (F5) i isprobajte!

8. POGLAVLJE: WINDOWS FORMS

Preostaje nam još dodati funkcionalnost drugom gumbu, nazvanom `button2`. Kôd koji ćemo kod njega dodati nakon dvoklika je sljedeći:

```
Close();
```

Taj komadić kôda nalaže formi u kojoj se nalazi da se zatvori, a pošto je naša forma osnovna (i jedina) forma naše aplikacije, zatvaranjem te forme zatvorit će se i cijela aplikacija. Uočite da prije metode `Close` ne navodimo kontrolu na koju se ona odnosi – stoga što se odnosi na formu.

Ukoliko aplikaciju želimo zatvoriti iz forme koja nije osnovna, koristit ćemo sljedeći kôd:

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

Ovaj primjer poslužit će nam da malo bolje upoznamo kontrole koje u njemu koristimo, njihova svojstva, događaje i metode.

Vizualna svojstva forme

Želite li da vam aplikacije budu dosadne, sive, jednolične i nimalo atraktivne? Trebali biste, jer prozorska aplikacija nije mjesto za pokazivanje umjetničkih sklonosti, no ipak ćemo vam pokazati kako izmijeniti neke vizualne karakteristike forme. Napominjemo još jednom – nemojte pretjerivati – aplikacije moraju biti jednostavne, oku ugodne i pregledne.

Dio vizualnih svojstava koje postavimo formi naslijedit će i sve kontrole koje ona sadrži. Primjerice, promijenimo li formi svojstvo za pozadinsku boju, nju će naslijediti i gumbi.



Boje i pozadina

Formi možemo mijenjati boju pozadine. To radimo mijenjajući svojstvo `BackColor` kojem valja pridružiti vrijednost tipa `System.Drawing.Color`. Njemu možemo pridružiti boje iz tri svojevrсна “spremišta” koje Visual Studio nudi u padajućem izborniku boja, no mi ćemo se upoznati s njima na nešto dubljem nivou pošto se koriste na mnogo mjesta, ne samo u ovoj kontroli nego i ostalima. Prvo “spremište” je skupina sistemskih boja u Windowsima koje su zapisane u klasi `System.Drawing.SystemColors`. Drugim riječima, kada napišemo sljedeću vrijednost:

```
System.Drawing.SystemColors.Control
```

dobit ćemo boju koja je zapisana u postavkama Windowsa kao *defaultna* boja za površinu prozora u *apletu* “Display Properties”. Na taj način poštujemo postavke korisnika koje je podesio

III. DIO: DIJELOVI .NET-A

na nivou operativnog sistema. Sve kontrole inicijalno imaju podešene boje na ovaj način i bez zaista jakog razloga ne biste to trebali dirati.

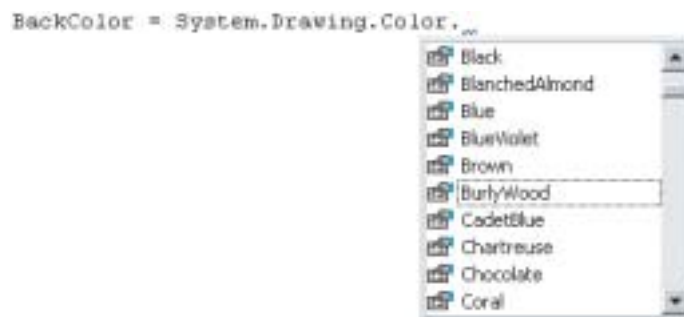


Svako svojstvo kontrole određenog je tipa i kod pridruživanja vrijednosti moramo joj pridružiti vrijednost upravo tog tipa. Neka svojstva su numeričkih vrijednosti (najčešće *int*), druga logičkih (*boolean*), a dobar dio svojstava je nekog posebnog tipa definiranog baš za to svojstvo. Na ovo valja paziti samo kod pridruživanja vrijednosti u kodu, a ako to radite pomoću Visual Studijevog sučelja, on će se sam za to pobrinuti.

Međutim, postoji mogućnost da želite podesiti neku konkretnu boju, nevezanu uz postavke operativnog sustava. Velikom broju boja možete pristupiti preko pamtljivih imena ispred kojih treba navesti ime klase u kojoj su definirane. Evo nekoliko primjera:

```
System.Drawing.Color.Black;
System.Drawing.Color.White;
System.Drawing.Color.Blue;
System.Drawing.Color.BurlyWood;
```

Slika 8-4:
IntelliSense u akciji – izbor boja



Dakako, imena boja ne morate pamtit. Dovoljno je u kodu napisati ime klase (*System.Drawing.Color*) i otipkati znak točke – IntelliSense će se pobrinuti za ostalo.

Treći način definiranja boja predviđen je za slučaj kada ne postoji predefinirano ime za boju koju želite koristiti. U tom slučaju, valja znati njenu RGB-vrijednost i napisati je u sljedećem obliku:

8. POGLAVLJE: WINDOWS FORMS

```
System.Drawing.Color.FromArgb(255, 128, 0);
```

RGB-vrijednosti boja sastoje se od tri broja. Prvi označava udio crvene boje (*red*), drugi zelene (*green*), a treći plave (*blue*). Brojevi moraju biti tipa *byte*, što znači da su u rasponu od 0 do 255. Vrijednost (0, 0, 0) predstavlja crnu, a (255, 255, 255) bijelu boju. Kratica RGB dolazi od prvih slova engleskih naziva boja.



Osim pozadine, boja se na formi koristi i za svojstvo `ForeColor`. Ono služi za definiranje boje teksta i grafike na formi. Kako forma sama po sebi ne sadrži nikakav tekst, do izražaja dolazi nasljeđivanje svojstava koje smo maloprije spomenuli – boja koju definirate svojstvu `ForeColor` neće biti aplicirana na formu, nego na sve kontrole koje se na njoj nalaze, osim onih koje za to svojstvo imaju definiranu drugu boju.

Kao primjer evo kôda koji valja ubaciti u funkciju vezanu uz događaj klika na gumb (`button1_Click`), i koji će učiniti da forma promijeni boju nakon klika na gumb:

```
BackColor = System.Drawing.Color.BlueViolet;
```

Osim boje, u pozadinu možemo smjestiti i sliku. To se radi postavljajući svojstvo `BackgroundImage`, za koje vrijedi isto upozorenje kao i za mijenjanje boja – samo ako morate.

Pozicija i veličina forme

Svojstvo `Location` određuje gdje će forma na ekranu biti pozicionirana. Ako, kao u našem primjeru, aplikacija ima samo jednu formu, onda ovo svojstvo određuje poziciju u odnosu na cijeli ekran. S druge strane, ukoliko se forma nalazi unutar neke druge forme (slučaj kakav ćemo upoznati nešto kasnije u poglavlju), onda se svojstvo `Location` odnosi relativno na tu roditeljsku formu. Ista je stvar i s kontrolama – koordinate u svojstvu `Location` odnose se na površinu forme na kojoj se kontrola nalazi.

Lokaciju ćemo najlakše definirati pomoću sučelja Visual Studija, no ako osjetite potrebu za promjenom ili čitanjem informacije o lokaciji forme u kodu, tom ćete svojstvu pristupiti ovako:

```
int kooX = Location.X; // čitanje koordinate X
int kooY = Location.Y; // čitanje koordinate Y

// premještaj forme na lokaciju 100, 200
Location = new System.Drawing.Point(100, 200);
```

III. DIO: DIJELOVI .NET-A

Primjećujete kako je repozicioniranje forme (ili kontrole općenito) prilično složeno. To je stoga što je svojstvo `Location` tipa `System.Drawing.Point`. Taj tip podataka je vrijednosni tip, što znači da vraća samu vrijednost, a ne i referencu na što se ona odnosi. Zato nije moguće napisati:

```
Location.X = 100; // pogrešno!
```

Stoga postoje dva dodatna svojstva koja nam omogućavaju jednostavnije pozicioniranje formi. Sljedeći primjer ima isti efekt kao i zadnja linija prošlog (ispravnog) primjera:

```
Left = 100;
Top = 200;
```

Slična je priča i sa svojstvima koja određuju veličinu forme. Pristupanje i mijenjanje kroz svojstvo `Size` slično je primjeru sa svojstvom `Location`:

```
int velX = Size.Width; // čitanje širine
int velY = Size.Height; // čitanje visine

// određivanje veličine forme na 500 x 600 piksela
Size = new System.Drawing.Size(500, 600);
```

Jednostavnije pisano, zadnji bi red izgledao ovako:

```
Width = 500;
Height = 600;
```



Osim spomenutih svojstava, postoje još svojstva `Right` i `Bottom`. Svojstvo `Right` jednako je zbroju svojstava `Left` i `Width`, dok je svojstvo `Bottom` jednako zbroju svojstava `Top` i `Height`. Ti odnosi među svojstvima su stalni, pa ako, primjerice, smanjite vrijednost `Height` za sto, i vrijednost `Bottom` će se smanjiti za isto toliko.

Pokazivač miša

Pokazivača miša korisnici računala prihvaćaju zdravo za gotovo. Sasvim je logično da on u nekim situacijama bude u obliku strelice, ponekad u obliku ruke, a ponekad u obliku za pisanje teksta. Naravno, vi to u svom programu možete kontrolirati, pa i promijeniti izgled pokazivača miša na vašoj formi.

8. POGLAVLJE: WINDOWS FORMS

To možete učiniti mijenjajući svojstvo `Cursor` odnosno pridružujući mu vrijednosti iz klase `System.Windows.Forms.Cursors`. Sučelje Visual Studija, kao i kod boja, nudi padajući izbornik za promjenu za vrijeme dizajniranja aplikacije, a ako to želite napraviti za vrijeme izvršavanja aplikacije, koristit ćete izraz poput ovoga:

```
Cursor = System.Windows.Forms.Cursors.WaitCursor;
```

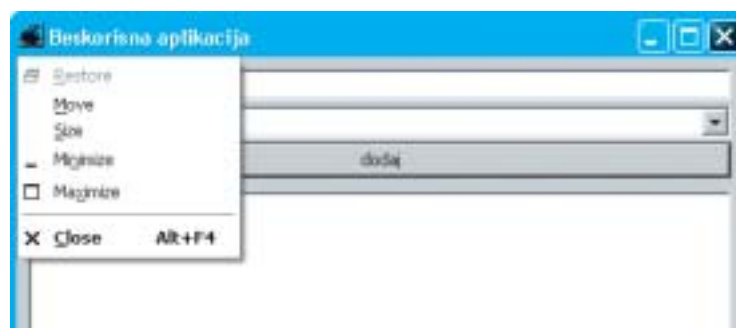
Kako na početku datoteke uključujemo *namespace* `System.Windows.Forms`, gornji redak možemo pisati i kraće:

```
Cursor = Cursors.WaitCursor
```

Zaglavlje forme

Svaki standardni prozor ima zaglavlje u kojem se nalaze ikona i naslov prozora s lijeve te kućice za minimiziranje, maksimiziranje, normalno stanje prozora i njegovo gašenje s desne strane. Pogledajte bilo koju klasičnu prozorsku aplikaciju i znat ćete o čemu pričamo.

Sva je ta svojstva moguće mijenjati odnosno utjecati na njihovo postojanje. Ikonu pridružujemo mijenjanjem svojstva `Icon`, na sličan način kao i pozadinsku sliku, dok je naslov prozora smješten u svojstvu `Text`.



Slika 8-5:
Zaglavlje forme (slijeva nadesno) – ikona forme (s odgovarajućim padajućim izbornikom), naslov, ikone za minimiziranje, maksimiziranje i zatvaranje

Dostupnost kućica za minimiziranje i maksimiziranje podešavamo pomoću svojstava `MinimizeBox` i `MaximizeBox` (pridružujemo im vrijednosti tipa *boolean*), a na isti način uključujemo odnosno isključujemo kućicu s upitnikom (za dozivanje sustava pomoći; svojstvo `HelpButton`). Zanimljivo je da kod svojstava vezanih uz minimiziranje i maksimiziranje forme ne isključujemo samo kućice

III. DIO: DIJELOVI .NET-A

u zaglavlju, nego općenito dozvoljavamo ili nedozvoljavamo minimiziranje odnosno maksimiziranje forme. Naime, osim preko tih kućica, te je radnje moguće izvršiti preko izbornika koji se otvara klikom na ikonu prozora ili desnim klikom na ime prozora u *taskbaru*.

Želite li sve muhe ubiti jednim udarcem, iskoristite svojstvo `ControlBox`. Postavljanjem vrijednosti tog svojstva na *false* ugasit ćete sve “sličice” u zaglavlju (ikonu i sve kućice s desne strane), a ostaviti samo naslov prozora.

Ponašanje i izgled forme

Nekoliko svojstava utječe na izgled forme. Tu treba spomenuti svojstvo `FormBorderStyle` koje definira koji tip obruba će forma imati. Osim vizualnog prikaza forme, kroz to svojstvo definiramo i hoće li korisnik programa moći mijenjati veličinu forme (*resize*) razvlačenjem ruba prozora. U tablici 8-1 možete vidjeti vrijednosti (pobrojane tipove) koje to svojstvo može poprimiti.

Tablica 8-1:
Moguće vrijednosti za svojstvo `FormBorderStyle`

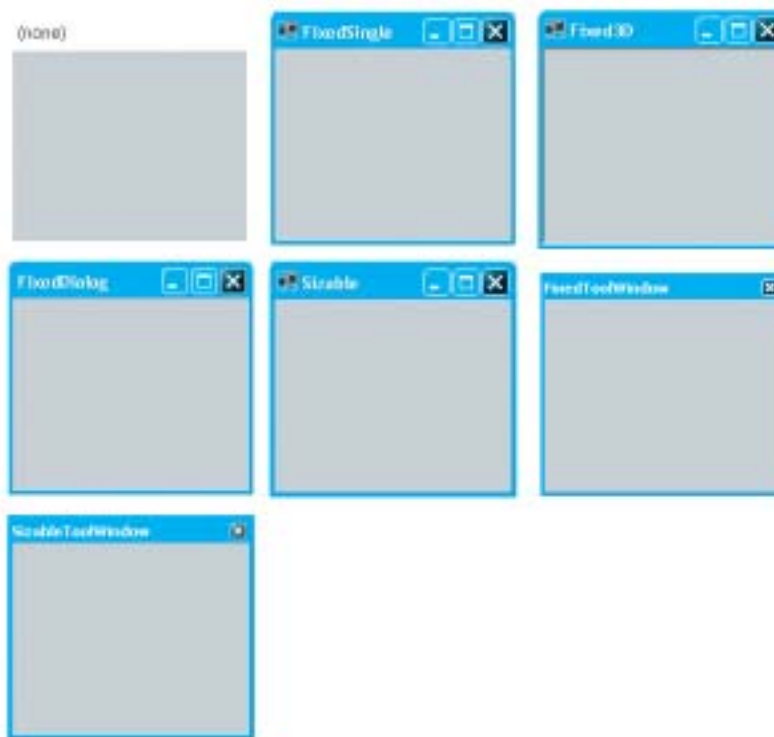
Vrijednost	Opis
<code>FormBorderStyle.Fixed3D</code>	fiksni trodimenzionalni rub
<code>FormBorderStyle.FixedDialog</code>	debeli fiksni rub u stilu dijaloškog okvira
<code>FormBorderStyle.FixedSingle</code>	tanki fiksni rub, debeo samo jedan piksel
<code>FormBorderStyle.FixedToolWindow</code>	fiksni rub tipa pomoćnog prozora (engl. <i>tool window</i>), koji se ne pojavljuje u <i>taskbaru</i> niti kada korisnik pritisne Alt + Tab
<code>FormBorderStyle.None</code>	bez ruba
<code>FormBorderStyle.Sizable</code>	trodimenzionalni rub; prozoru je moguće mijenjati dimenzije
<code>FormBorderStyle.SizableToolWindow</code>	promjenjivi pomoćni prozor, koji se ne pojavljuje u <i>taskbaru</i> niti kada korisnik pritisne Alt + Tab .

Kao što piše u tablici, pomoćni prozori se ne pojavljuju u *taskbaru*. Međutim, ako želimo da se i prozor nekog drugog tipa tamo ne pokazuje, možemo to napraviti mijenjajući vrijednost svojstvu `ShowInTaskbar`, za vrijeme dizajniranja forme ili za vrijeme izvršavanja aplikacije.

Pomoću svojstva `StartPosition` određujemo gdje će se forma prilikom aktivacije pojaviti. Ukoliko pridružimo vrijednost `FormStartPosition.Manual`, forma će se smjestiti na parametre određene svojstvom `Location`. `FormStartPosition.CenterScreen` nalaže prikaz forme na centru ekrana, `FormStartPo-`

8. POGLAVLJE: WINDOWS FORMS

sition.CenterParent u centru roditeljske kontrole (o tome nešto kasnije), a FormStartPosition.WindowsDefaultLocation i FormStartPosition.WindowsDefaultBounds na mjesto definirano u Windowsima, s razlikom da potonji ignorira veličinu prozora definiranu svojstvom Size.



Slika 8-6:
*Usporedba različitih
stilova formi (svojstvo
FormBorderStyle)*

Formi možemo odrediti da se otvara u maksimiziranom odnosno minimiziranom stanju, pa svojstvo iz prethodnog odlomka neće imati smisla. To će se dogoditi ako svojstvu WindowState dodelimo vrijednost FormWindowState.Minimized odnosno FormWindowState.Maximized. To svojstvo možemo mijenjati i za vrijeme izvršavanja aplikacije, pa tako možemo našem sveprisutnom događaju klika na gumb dodati liniju koja slijedi, i na taj način maksimizirati formu:

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

Konačno, prozoru možemo uključiti svojstvo TopMost koje će ga držati na vrhu svih prozora, bez obzira na to što nije trenutno aktivan prozor. Ta je mogućnost korisnicima poznata pod imenom "Always on top".

Kako bismo isprobali tu mogućnost, dodat ćemo sljedeću liniju kôda (znate već gdje):

III. DIO: DIJELOVI .NET-A

```
TopMost = ! (TopMost);
```

Ta će linija omogućiti da pritiskom na gumb uključujemo i isključujemo svojstvo forme da bude iznad svih prozora ne samo svoje, već i ostalih aplikacija. Ukoliko je to svojstvo uključeno (*true*), znak uskličnika će ga pretvoriti u neistinitu logičku vrijednost (*false*) i zatim tu vrijednost pridružiti svojstvu. I obrnuto.

Prozirne i šuplje forme

Iako ih rijetko susrećemo u praksi (što nije nužno loše), prozirne i šuplje forme vrlo je jednostavno napraviti. Prozirnost forme definiramo parametrom *Opacity*, kojem navodimo postotnu vrijednost prozirnosti forme. *Defaultna* vrijednost je 100%, što znači da forma neće biti prozirna.

Šupljinu u formi određujemo pomoću boje. Naime, svojstvu *TransparencyKey* pridružujemo neku boju, a ono se brine da svako pojavljivanje te boje u formi rezultira – rupom. Kroz tu rupu u prozoru možete vidjeti i kliknuti ono što bi inače bilo prekriveno prozorom. Postavimo li vrijednost tog svojstva na bijelu boju, naš će primjer dobiti rupu na tekstualnom polju za upis, padajućem izborniku i listi.

Slika 8-7:
Forma nekonven-
cionalnog oblika
napravljena modifikaci-
jom našeg primjera



Prozirne forme rijetko imaju smislen razlog postojanja, no šupljima se već može naći korisnih implementacija. Naime, one se često koriste za kreiranje prozora nekonvencionalnih oblika, poput prozora sa zaobljenim rubovima ili forme u obliku kruga.

8. POGLAVLJE: WINDOWS FORMS

Na slici 8-7 možete vidjeti jednu takvu formu, napravljenu na temelju primjera Beskorisne aplikacije. Recept je sljedeći: prvo valja napraviti sliku koju ćemo staviti u pozadinu forme. Jedna od boja na toj formi (po mogućnosti neka kričava, koja se sigurno neće pojavljivati na formi) bit će zamijenjena “rupom”, tako da ćete nju najvjerojatnije staviti na rub slike. Ostatak slike, ono što će biti vidljivo, obojite nekom drugom bojom, najbolje sivom, koja se i inače koristi za pozadinu prozora.

Tako pripremljenu sliku pridružite svojstvu `BackgroundImage`. Zatim svojstvu `TransparencyKey` pridružite onu kričavu boju sa slike kako bi taj dio prozora postao šupalj. Konačno, svojstvu `FormBorderStyle` pridružite vrijednost `None` kako biste uklonili rub forme i zaglavlje, koji bi uništili cijeli posao.

Olakšajmo potrebnima!

K ako bi i osobe s invaliditetom mogle koristiti računala, postoje posebni dodaci za operativne sustave koji im u tome pomažu. Neki od tih dodataka nalaze se i u Windowsima, a neke je potrebno naknadno instalirati.

Ta se dodatna programska podrška ne oslanja samo na operativni sustav, već i na aplikacije. Tako i vaša aplikacija može biti prilagođena za korištenje od strane osoba s invaliditetom, i to podešavajući tri jednostavna svojstva koja se nalaze na svim vizualnim kontrolama.

Svojstvo `AccessibleName` treba sadržavati kratko ime koje korisnika može asocirati na funkciju kontrole. Nešto duži opis kontrole valja upisati u svojstvo `AccessibleDescription`, dok kao vrijednost svojstva `AccessibleRole` treba izabrati ulogu te kontrole. U većini slučajeva ponje svojstvo nije potrebno mijenjati s početne vrijednosti `Default` jer će kroz tu vrijednost sistem za svaku kontrolu vraćati njenu pravu ulogu.

Odnos kontrola i forme

Kao što smo već više puta ponovili, forma može sadržavati razne kontrole. One se na formi ne nalaze samo vizualno, nego to pravilo slijedi i objektni model. Sve se kontrole, naime, nalaze u kolekciji `Controls`. Primjerice, želimo li nekoj od njih promijeniti svojstvo `Text`, to možemo učiniti i ovako:

```
Controls[0].Text = "novo svojstvo Text";
```

Kolekcija `Controls` odnosi se na formu, no ime forme nije potrebno navoditi jer se sav kôd nalazi u klasi (formi) na koju se odnosi. Indeks “0” označava prvu kontrolu na formi, no često je teško

III. DIO: DIJELOVI .NET-A

znati koja je to. Naime, prva kontrola je ona koja se prva kreira, što u praksi ne znamo iako možemo vidjeti u skrivenom dijelu kôda (regija Windows Form Designer generated code). To je ona kontrola nad kojom je prva pozvana metoda Controls.Add. U praksi to izgleda ovako:

```
this.Controls.Add(this.button2); // kontrola s indeksom 0
this.Controls.Add(this.button1); // kontrola s indeksom 1
this.Controls.Add(this.listBox1); // kontrola s indeksom 2
```

Ovakvo korištenje kontrola je prekomplikirano, pa postoji direktno referenciranje kontrola koje smo već susreli:

```
button2.Text = "novo svojstvo Text"
```

Zanimljivo je da osim što preko forme možete doći do kontrola, i preko kontrola možete doći do forme kojoj ona pripada. Evo kako preko kontrole promijeniti svojstvo koje pripada formi:

```
button2.TopLevelControl.Text = "Novi naslov forme";
```

Dakako, istu je stvar moguće dobiti osjetno kraćim izrazom, no samo smo željeli pokazati kako su forma i njezine kontrole povezane. Osim gornjega, postoje još perverziji primjeri, kao što je sljedeći koji preko kontrole pristupa formi da bi promijenio svojstvo tog istoj kontroli:

```
button2.TopLevelControl.Controls[0].Text = "Novi natpis na button2";
```

Nakon što smo objasnili objektnu povezanost forme i njezinih kontrola, red je da se pozabavimo nešto manje apstraktnim stvarima.

Pozicija, veličina i skrivanje kontrola

Pozicija kontrola relativna je u odnosu na unutarnji dio forme (znači, ne računaju se rubovi i zaglavlje forme). Drugim riječima, ako neka kontrola stoji na poziciji 100, 200, to znači da je ona 100 piksela desno i 200 piksela dolje od unutarnjeg gornjeg lijevog ruba. U većini slučajeva nećete u kodu mijenjati pozicije i veličine kontrola već ćete ih vizualno postaviti u dizajnerskom načinu. Ipak, ukoliko vam zatreba, sintaksa je jednaka kao kod pozicioniranja forme (jer, sjetimo se, i forma je kontrola):

```
int kooX = button1.Location.X; // čitanje koordinate X
int kooY = button1.Location.Y; // čitanje koordinate Y

// premještaj kontrole button1 na lokaciju 100, 200
button1.Location = new System.Drawing.Point(100, 200);
```

8. POGLAVLJE: WINDOWS FORMS

```
// drugi način premještanja kontrole
button1.Left = 100;
button1.Top = 200;
```

Ista je stvar i s veličinom kontrola:

```
int velX = button1.Size.Width; // čitanje širine
int velY = button1.Size.Height; // čitanje visine

// određivanje veličine kontrole na 200 x 40 piksela
button1.Size = new System.Drawing.Size(200, 40);

// drugi način određivanja veličine kontrole
button1.Width = 200;
button1.Height = 40;
```

Kontrole na formi možete i skrivati. To činite postavljanjem svojstva `Visible` na `true` (kontrola će biti vidljiva) odnosno na `false` (kontrola će biti nevidljiva). Imajte na umu da ovo utječe isključivo na vizualnu karakteristiku kontrole – ona i dalje postoji i sva njezina svojstva se mogu normalno koristiti.

Ukoliko vam se ne da natezati s koordinatama i veličinom kontrole, možete joj reći da bude zalijepljena cijelom dužinom uz neki od rubova. To radimo svojstvom `Dock`, koje možemo postaviti na vrijednosti `None`, `Top`, `Bottom`, `Left` i `Fill`.

Razvlačenje formi

Korisnici su navikli da veličinu prozora (forme) mogu prilagoditi svojim željama i potrebama. Međutim, ako netko razvuče formu iz našeg primjera, dobit će samo velik, neiskorišten prostor.

Ne zaboravite, da bi forma mogla biti razvlačena, mora biti adekvatno podešeno svojstvo `FormBorderStyle`!



Da se to ne bi događalo, kontrole imaju svojstvo `Anchor`. Ono je inicijalno postavljeno na vrijednosti (`Top` | `Left`). Zbog takvih postavki događa se da sve kontrole prilikom razvlačenja ostanu u gornjem (`Top`) lijevom (`Left`) uglu forme. Želimo li da sve budu vezane uz donji desni ugao, svima ćemo svojstvo `Anchor` postaviti na vrijednosti (`Bottom` | `Right`). Ako navedemo dvije suprotne vrijednosti, prim-

III. DIO: DIJELOVI .NET-A

jerice (Left | Right), onda kontrola neće mijenjati lokaciju, nego veličinu – bit će razvučena u tom smjeru.



Slika 8-8:
Beskorisno razvučena
Beskorisna aplikacija



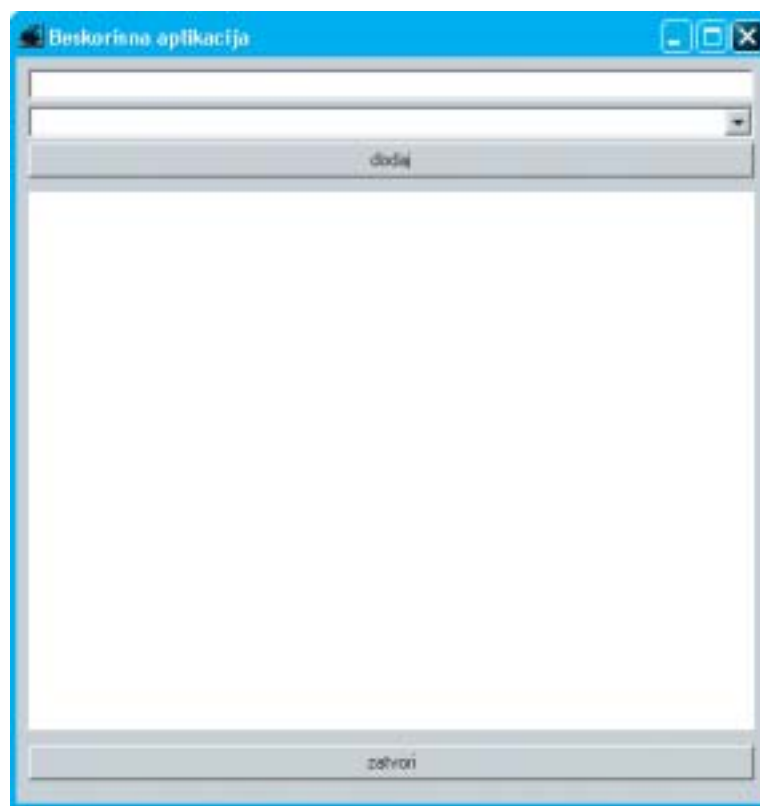
Prvo ćemo se pozabaviti vodoravnim razvlačenjem. Ako korisnik razvuče formu u širinu, želimo da sve kontrole također budu razvučene u širinu tako da razmak između kontrole i desnog ruba bude uvijek isti. Zato ćemo svim kontrolama svojstvo Anchor podesiti na vrijednosti (Top | Left | Right). Time smo odredili da se prilikom razvlačenja forme u širinu i sve kontrole na njoj proporcionalno prošire.



Želite li neko svojstvo promijeniti na više kontrola odjednom, označite sve takve kontrole (držite tipku Ctrl prilikom označavanja) i onda izmijenite željeno svojstvo. Promjena će biti aplicirana na sve kontrole koje su bile označene. Pripazite da sve kontrole imaju svojstvo koje želite mijenjati!

8. POGLAVLJE: WINDOWS FORMS

Preostaje nam rješavanje okomitog razvlačenja. Ukoliko tu primijenimo isti princip, kontrole će se početi razvlačiti jedna preko druge i sučelje će biti uništeno. Zato trebamo odrediti jednu centralnu kontrolu koja će zauzeti višak površine, dok će ostale samo mijenjati poziciju, a ne i visinu. Takav je princip apliciran kod svih aplikacija u Windowsima – razvlačite li prozor Microsoft Worda, ulogu centralne kontrole imat će središnji, bijeli dio u koji unosimo tekst.



Slika 8-9:
Beskorisna aplikacija s
pravilno podešenim
svojstvom Anchor

Pravilo koje vrijedi u većini slučajeva jest – najveća kontrola dobiva ulogu centralne. U našem primjeru to je kontrola listBox1. Dalje je jednostavno – centralnoj kontroli dodjeljujemo sve četiri vrijednosti svojstva Anchor – (Top | Left | Right | Bottom). Kontrole koje se nalaze iznad nje dobivaju vrijednosti (Top | Left | Right), dok onima ispod nje pridružujemo (Left | Right | Bottom). Rezultat možete vidjeti na slici 8-9. (Preporučujemo vam da ovo svojstvo podešavate na kraju dizajniranja korisničkog sučelja, jer svoje ponašanje pokazuje i u dizajnerskom načinu, što može biti smetnja.)

III. DIO: DIJELOVI .NET-A

Razvlačenje formi u aplikacijama je zgodna stvar, no može postojati granica do koje želite korisniku dozvoliti smanjivanje (ili povećavanje) forme. Primjerice, forma veličine 10x10 piksela nema nikakvog smisla, jer kontrole neće biti niti vidljive, a kamoli upotrebljive. Stoga vam biblioteka Windows Forms omogućava postavljanje gornje i donje granice dimenzija forme. One se kriju iza svojstava `MaximumSize` i `MinimumSize` i istog su tipa kao i svojstvo `Size`. Ukoliko su ta svojstva postavljena na (0, 0), granice neće postojati.



Osim na razvlačenje, svojstvo `MaximumSize` utječe i na maksimiziranje forme. Drugim riječima, ukoliko maksimizirate formu koja ima postavljeno svojstvo `MaximumSize`, ona će biti povećana do granice definirane u tom svojstvu, a ne preko cijelog ekrana.

Gumbi potvrde i poništenja

Ako ste se malo više igrali s našim primjerom, onda ste sigurno poželjeli da nakon upisa i odabira parametara ne morate u ruke uzimati miša, nego da umjesto klika na gumb možete istu stvar napraviti pritiskom tipke `Enter` na tipkovnici. Ništa lakše! Treba u svojstvima forme potražiti svojstvo `AcceptButton` i iz padajuće liste izabrati gumb koji želimo da bude pritisnut kada korisnik klikne tipku `Enter`. U našem slučaju to će biti `button1`.

Istu stvar možete napraviti i sa svojstvom `CancelButton`, jedino što on neće reagirati na tipku `Enter`, već na tipku `Esc`. U našem primjeru tom svojstvu pridružen je gumb `button2`, tako da se pritiskom tipke `Esc` zatvori forma i završava program.

Tipkovnička šetnja sučeljem

Osim tipaka `Enter` i `Esc`, vrlo često korištena tipka za navigaciju poljima za unos je i tipka `Tab` odnosno kombinacija `Shift + Tab`. Njihovu namjenu već znate – prelazak u sljedeće odnosno prethodno polje na stranici – a kako ih implementirati u vlastiti program, saznat ćete sada.

Zapravo, neprecizno je govoriti o implementaciji, jer je samim kreiranjem sučelja cijela stvar već funkcionalna. Vi možete i morate podesiti stvar da radi u skladu s očekivanjima korisnika. Uzmimo za primjer našu Beskorisnu aplikaciju – nije logično da nas nakon tekstualnog polja za upis tipka `Tab` prebaci na gumb ili listu. Trebamo postići da se prebacivanje vrši na vizualno prvu sljedeću kontrolu kako bi ponašanje sučelja bilo intuitivno i očekivano. To radimo mijenjajući vrijednost svojstva `TabIndex` svakoj kontroli. Kontrolu za koju želimo da prva ima fokus, odmah nakon pokretanja programa, dodijelit ćemo vrijednost 0. Sljedećoj ćemo pridružiti vrijednost 1, onoj nakon nje 2 i tako dalje.

8. POGLAVLJE: WINDOWS FORMS

Ukoliko ne dodijelite vrijednosti svojstvima `TabIndex`, tipka `Tab` koristit će onaj redosljed kojim su kontrole dodavane u kolekciju `Controls`. To u većini slučajeva nije dobro rješenje, pa ovoj funkcionalnosti svakako posvetite pažnju.



Naravno, ne može se do svih kontrola doći tipkom `Tab`. Neke od njih za tim jednostavno nemaju potrebe jer ne pružaju nikakvu interaktivnu funkcionalnost (najbolji primjer za takve kontrole su one tipa `Label`, o kojima ćemo nešto kasnije), dok nekima i sami možete isključiti tu funkcionalnost. Tome služi svojstvo `TabStop` kojem treba pridružiti vrijednost tipa *boolean*. Sve kontrole to svojstvo imaju inicijalno postavljeno na *true*, što znači da im je pristup tipkom `Tab` omogućen.

Gumbi

Nakon sage o formama, vrijeme je da se поближе pozabavimo kontrolama koje smo koristili u primjeru Beskorisne aplikacije. Jedna od najvažnijih kontrola je ona tipa `Button`. Vjerujemo da vam funkciju gumba u aplikaciji ne moramo objašnjavati, no zanimljivo je proći neka njegova svojstva, kako biste ga u svojim aplikacijama mogli bolje iskoristiti.

Svojstva koja spominjemo kod neke kontrole mogu vrijediti i za druge kontrole. Priručno pravilo glasi: ako se svojstva isto zovu, onda imaju istu ili vrlo sličnu funkciju. Kod nekih ćemo kontrola spomenuti svojstva koja "dijeli" s već spomenutim kontrolama, no kod većine nećemo, jer bi zbog velikog broja svojstava takva nabrojavanja bila nepregledna i nepraktična. Popis svih svojstava dostupnih kod neke kontrole možete pronaći u dokumentaciji ili u sučelju Visual Studija.



Odluku o stilu gumba morate donijeti na početku izrade aplikacije. Naime, vrlo je važno da vam svi gumbi budu istog stila, kako bi aplikacija dobila na preglednosti. Stil gumba mijenjate mijenjajući svojstvo `FlatStyle`, kojem možete dodijeliti četiri vrijednosti (iz pobrojanog tipa `System.Windows.Forms.FlatStyle`): `Standard`, `Flat`, `Popup` i `System`. Usporedbu prvih triju tipova možete vidjeti na slici 8-10, dok četvrti predstavlja jedan od njih, ovisno o postavkama sistema.

Svaki gumb ima natpis kojim korisniku daje do znanja čemu služi. Taj se natpis mijenja svojstvom `Text`. Poziciju natpisa u gumbu možete podesiti mijenjajući svojstvo `TextAlign`. Na taj način natpis možete smjestiti u sredinu gumba (inicijalna postavka), uz neki od rubova ili u neki od kutova. Tip i veličinu slova kojima je ispisan natpis možemo mijenjati preko svojstva `Font` iako to nije uobičajeno.

III. DIO: DIJELOVI .NET-A

Slika 8-10:
*Usporedba triju stilova
gumba u tri različita sta-
nja – izaberite koji želite,
samo budite dosljedni!*

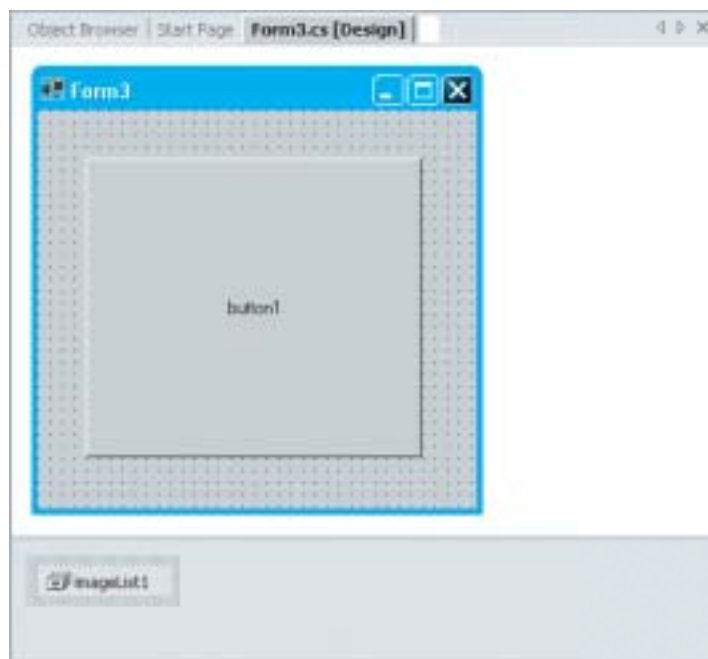


Osim natpisa, gumb može imati i sličicu. Nju postavljamo tako da svojstvu Image dodijelimo sliku, na isti način kao što smo radili s formom. Poziciju slike na gumbu, slično kao što smo radili s tekстом, mijenjamo svojstvom ImageAlign.

Lista slika

Međutim, često je potrebno na istom gumbu imati više slika. Primjerice, jednu kada je gumb u stanju mirovanja, drugu kada se nad njime nalazi pokazivač miša i treću kada je pritisnut. Kako to napraviti?

Slika 8-11:
*Nevizualne kontrole
prikazuju se izvan
forme.*

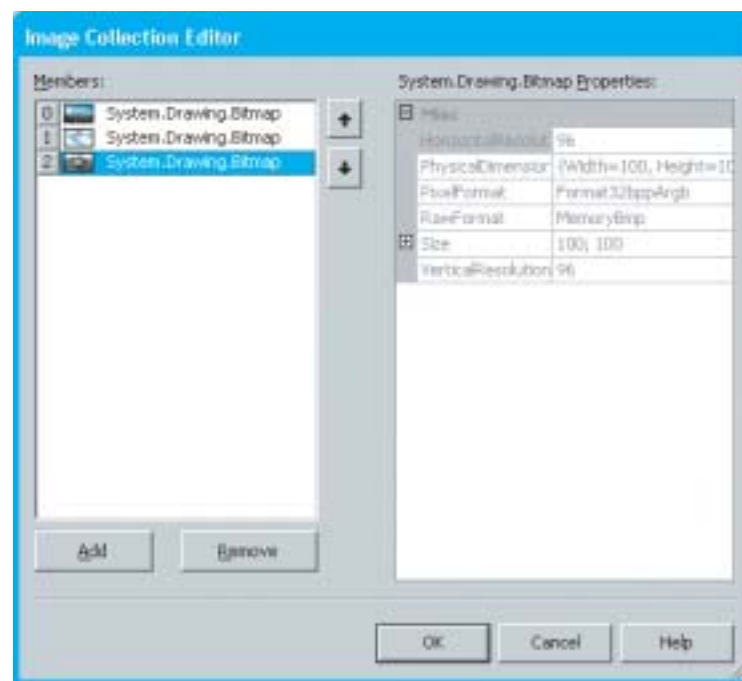


8. POGLAVLJE: WINDOWS FORMS

Da bismo to postigli, moramo na formu dovući kontrolu tipa ImageList. Ta kontrola ne spada među vizualne kontrole, pa se neće pojaviti na samoj formi, nego u području ispod nje (vidi sliku 8-11).

Kontrola ImageList predstavlja svojevršno spremište slika. Svaka slika koju dodamo u tu kontrolu bit će sastavni dio aplikacije i moći ćemo je koristiti u mnogim slučajevima, uključujući i naš primjer s gumbom koji se upravo spremamo napisati.

U listu sliku treba smjestiti onoliko slika koliko vam treba, a u našem slučaju će to biti tri slike. Bitno je da pripazite da slike budu istih dimenzija kao i gumb – ne zato što drugačije ne bi radi- lo, nego zato što će tako ljepše izgledati.



Slika 8-12:
Pomoćni prozor za
dodavanje slikovnih
datoteka u listu slika

Slike dodajete preko svojstva Images. Pritiskom na gumbić s tri točkice otvorit će vam se prozor (slika 8-12) u kojem ćete slike dodavati klikom na gumb Add. Taj prozor krije i jednu vrlo važnu stvar – indekse slika u kolekciji. Kao i kod svih ostalih kolekcija, prva slika ima indeks 0. Zapamtite koja slika ima koji indeks, kako biste ih kasnije znali referencirati. Najbolje je smisliti i slijediti neki (va- ma) logičan princip. Primjerice, mi ćemo staviti tri slike – prvu (indeks 0) za gumb u stanju miro- vanja, drugu za gumb ispod pokazivača miša i treću za pritisnut gumb.

Nakon što smo zatvorili pomoćni prozor, trebamo podesiti još neka svojstva liste slika. Nikako ne smi- jemo zaboraviti svojstvo ImageSize pomoću kojeg definiramo veličinu slika. Inicijalna vrijednost je

III. DIO: DIJELOVI .NET-A

16x16 piksela, što u praksi znači da će vaše slike, bez obzira na originalnu veličinu, biti reducirane na tu dimenziju. To, dakako, nije poželjno, pa stoga tu vrijednost podesite na dimenziju slika koje ste dodali.

Osim dimenzija možete podesiti dubinu (broj) boja svojstvom `ColorDepth` i prozirnost slike odabirom neke boje u svojstvu `TransparentColor`.

Vrijeme je da napravljenu listu slika povežemo gumbom. To činimo tako da među svojstvima gumba nađemo ono nazvano `ImageList`. Tamo će se u padajućoj listi pojaviti naša lista slika (*default-nog* imena `imageList1`) koju trebamo izabrati. Naravno, sučelje ne zna koju sliku iz liste želimo postaviti kao osnovnu, pa stoga u svojstvo `ImageIndex` trebamo upisati vrijednost indeksa slike. Radi se o svojstvu tipa *int*, no sučelje nam pomaže u obliku malih inačica slika, koje se kriju iza pojedinog indeksa kako bismo lakše izabrali. Mi odabiremo indeks 0, kako bismo prvu sliku postavili za osnovnu.

Događaji vezani uz miša

Za ostatak funkcionalnosti morat ćemo posegnuti za malo kôda. To ćemo učiniti tako da za određene događaje na gumbu vezemo funkcije koje će na njemu mijenjati sliku. Za početak, prilikom prelaska pokazivačem miša želimo da se prikaže slika s indeksom 1. U pomoćnom prozoru `Properties` prelazimo na popis događaja (ikona žute munje), pronalazimo događaj `MouseEnter`, dvoklikom i otvara nam se funkcija koja treba izgledati ovako:

```
private void button1_MouseEnter(object sender, System.EventArgs e)
{
    button1.ImageIndex = 1;
}
```

Događaj `MouseEnter` nastupa kada pokazivač miša uđe u vidljivo područje kontrole na koju se odnosi, u našem slučaju kontrole `button1`. Promjenom indeksa u svojstvu `ImageIndex`, automatski se mijenja i slika prikazana na gumbu.



Prilikom nastupa nekog događaja (primjerice `MouseEnter`), svaka kontrola ima i ugrađenu metodu (`OnMouseEnter`) koja će se automatski izvršiti. Uz nju, mi možemo napisati vlastitu funkciju vezanu uz taj događaj (koja će se automatski nazvati `ImeKontrole_Enter`). Uvijek se izvršavaju obje funkcije, i naša i ugrađena.

Kada se pokazivač miša makne s gumba, treba vratiti prvu sliku. Ovaj se puta vezemo na događaj `MouseLeave`:

8. POGLAVLJE: WINDOWS FORMS

```
private void button1_MouseLeave(object sender, System.EventArgs e)
{
    button1.ImageIndex = 0;
}
```

U slučaju pritiska tipke miša nastupa događaj `MouseDown`, a kada tipku otpustimo, nastupit će događaj `MouseUp`. Evo kôda koji valja zakačiti na te događaje:

```
private void button1_MouseDown(...)
{
    button1.ImageIndex = 2;
}

private void button1_MouseUp(...)
{
    button1.ImageIndex = 1;
}
```

Kad se pritisne tipka miša, prikazat će se treća slika (indeks 2) iz liste slika. Kada se tipka otpusti, prikazuje se druga slika (indeks 1). Zašto druga, pitate se, kad je prva osnovna slika? Zato što se u trenutku otpusta tipke pokazivač miša još uvijek nalazi nad gumbom, pa je potrebno vratiti onu sliku koja se tada prikazuje, a to je u našem slučaju druga slika.

Događaji koje smo upravo obradili postoje u svim vizualnim kontrolama; osim njih, postoje još dva događaja vezana uz miša i njegove dogodovštine na ekranu. Prvi među njima je `MouseMove` koji se događa svaki put kada se miš pomakne nad kontrolom. Kod korištenja tog događaja vrlo je važno imati na umu da može nastupiti izuzetno velik broj puta, posebno ako korisnik miša pomiče vrlo sporo ili kruži unutar jedne te iste kontrole. Smještanje čak i jednostavnog kôda u funkciju vezanu uz taj događaj može rezultirati značajnim trošenjem sistemskih resursa.

Da biste dobili dojam koliko često događaj `MouseMove` nastupa, povežite na njega sljedeću funkciju:

```
private void button1_MouseMove(...)
{
    Text += ".";
}
```

Ta će funkcija svaki put kada bude pozvana dodati jednu točku u naslov prozora. Primijetite kako će se brzo točke nakupiti!

Zadnji događaj iz paketa “mišjih” događaja je `MouseHover`, koji nastupa u slučaju da se korisnik neko kraće vrijeme zadrži nad kontrolom ništa ne radeći. Najbolji primjer tog događaja (iako se

III. DIO: DIJELOVI .NET-A

ne rješava pomoću njega) jest onaj mali, žuti pravokutnik koji se u brojnim aplikacijama pojavljuje nad ikonama u alatnoj traci da nam u nekoliko riječi opiše funkciju koja se krije iza ikone.

Polje za unos teksta (jedna linija)

Sljedeća kontrola na listi za proučavanje je TextBox. Kao što nezgrapni prijevod u naslovu kaže, radi se o kontroli koja korisniku omogućava upis niza znakova.



Kontrola TextBox koristi se i za jednolinijske i višelinijske upise teksta. Na ovom mjestu proučavamo osnovna svojstva vezana uz jednolinijsku funkcionalnost, dok ćemo se svojstvima vezanim uz tekstualno polje za upis s više linija baviti u sljedećem primjeru.

Osnovne stvari u radu s ovom kontrolom vidjeli smo u primjeru Beskorisna aplikacija. Znamo da se uneseni tekst sprema u svojstvo Text i kako mu se pristupa.

Sad ćemo se malo pozabaviti vizualnim karakteristikama polja za unos teksta. Za početak, moguće je postaviti tri stila okvira mijenjajući svojstvo BorderStyle. Dostupni su nam *defaultni* trodimenzionalni okvir (vrijednost Fixed3D), jednostavan okvir s rubom debelim jedan piksel (FixedSingle) i polje za unos bez okvira (None).

Tip i veličinu slova (svojstvo Font), kao i kod ostalih kontrola, možete mijenjati i kontrolama tipa TextBox, a da ne biste trebali ručno podešavati veličinu kontrole, postoji svojstvo AutoSize. Kada je njemu pridružena vrijednost *true*, visina kontrole automatski će se prilagoditi odabranom tipu slova.

Broj znakova koje korisnik unosi možete limitirati mijenjajući svojstvo MaxLength. Želite li da korisnik može unositi isključivo mala ili isključivo velika slova, svojstvo CharacterCasting postavite na jednu od vrijednosti iz pobrojanog tipa System.Windows.Forms.CharacterCasing (Lower ili Upper umjesto Normal).

Kada želite da korisnik ne vidi slova koja upisuje (primjerice, kod unosa lozinke), u svojstvo PasswordChar unijet ćete znak (samo jedan!) koji će se pokazivati na ekranu bez obzira na to koje je slovo pritisnuto. Uvriježilo se da taj znak bude zvjezdica (*).

Ponekad postoji potreba za onemogućavanjem unošenja ili mijenjanja teksta u polju. To, dakako, možete riješiti skrivanjem kontrole mijenjajući svojstvo Visible, no puno elegantnije i preglednije rješenje se krije iza svojstva ReadOnly. Ako je ono postavljeno na *true*, korisniku neće biti dozvoljena nikakva promjena u polju za upis, a cijelo će polje biti zasivljeno.

Događaji vezani uz promjene svojstava

U priči o poljima za unos teksta javlja se prilika da se malo više pozabavimo događajima koji nastupaju kad se neko svojstvo promijeni. Stvar je vrlo jednostavna – primjerice, u trenutku kada se promijeni vrijednost svojstva `Text`, nastupit će događaj `TextChanged`. Mi za njega, kao i za svaki drugi događaj, možemo vezati funkciju i tako dodati neku funkcionalnost.

Događaji koji nastupaju nakon promjene nekog svojstva ne postoje za sva svojstva. Koja su svojstva dostupna za koju kontrolu, potražite u dokumentaciji ili sučelju Visual Studija.



U sljedećem primjeru napraviti ćemo da svakom promjenom teksta u polju za upis promijenimo i ime forme:

```
private void textBox1_TextChanged(...)
{
    Text = textBox1.Text;
}
```

Padajuća lista

Ukoliko korisniku ne želite omogućiti slobodan upis neke informacije već želite da izabere iz liste ponuđenih opcija, koristit ćete kontrolu `ComboBox`. Njezino glavno svojstvo je `DropDownStyle`, kojim određujete izgled i ponašanje kontrole. To svojstvo tipa `System.Windows.Forms.ComboBoxStyle` inicijalno ima vrijednost `DropDown` koja osim izbora među ponuđenim opcijama nudi i mogućnost upisivanja vlastite vrijednosti. Ako ne želite korisniku dozvoliti slobodan unos, svojstvo `DropDownStyle` postavite na vrijednost `DropDownList`. Treća moguća vrijednost (`Simple`) rijetko se koristi. Po funkcionalnosti je slična prvoj, osim što ne postoji strelica za otvaranje padajuće liste, već kontrola na formi izgleda kao polje za unos teksta, a kroz ponuđene opcije korisnik se mora kretati strelicama na tipkovnici.

Već smo pokazali kako pomoću Visual Studijevog sučelja dodati opcije u padajuću listu (svojstvo `Items`). Sada nam preostaje pokazati kako opcije dodavati i mijenjati u kodu. Želimo li dodati dodatnu opciju, koristit ćemo sljedeću sintaksu:

```
comboBox1.Items.Add("nova opcija");
```

III. DIO: DIJELOVI .NET-A

Primjećujete – stvar izgleda isto kao i dodavanje nove stavke u listu koju smo koristili u primjeru Beskorisna aplikacija. Pažljiviji čitatelji petog poglavlja prepoznat će u svojstvu `Items` kolekciju *stringova*, s kojom možemo raditi sve što smo u tom poglavlju naučili – dodavati, brisati, mijenjati i prebrojavati. Evo nekoliko primjera:

```
// brisanje svih članova iz kolekcije
comboBox1.Items.Clear();

// dodavanje novog člana umetanjem na treću poziciju
comboBox1.Items.Insert(2, "Novi član kolekcije");

// brisanje člana s indeksom 2
comboBox1.Items.RemoveAt(2);

// brisanje člana s određenom vrijednošću
comboBox1.Items.Remove("Novi član kolekcije");
```

Padajuća lista može sadržavati neograničen broj članova, a koliko će ih odjednom biti vidljivo na ekranu ovisi o vrijednosti svojstva `MaxDropDownItems`. To, naravno, ne znači da ostali članovi neće biti dostupni za izbor, nego da ćemo, da bismo do njih došli, morati skrolati listu.

Kontrola padajuća lista omogućava i automatsko sortiranje svojih članova. Ukoliko želimo da budu sortirani, svojstvo `Sorted` postaviti ćemo na *true*. Imajte na umu da ovim svojstvom možete unijeti zbrku u indekse unutar kolekcije, jer, primjerice, posljednji dodani član neće nužno biti na posljednjem mjestu, već na onom kojem mu pripada u sortiranom nizu. Međutim, ukoliko promijenite nekome od članova ime, kolekcija neće automatski biti sortirana, već će član ostati na mjestu na kojem se nalazio. Želite li takvu kolekciju ipak sortirati, možete napraviti sljedeće:

```
comboBox1.Sorted = false;
comboBox1.Sorted = true;
```

Spomenuti trik ne biste smjeli prečesto koristiti jer je sortiranje kolekcije na taj način prilično sporo, a o sortiranju većih kolekcija na ovaj način nemojte niti razmišljati.

Kada korisnik odabere jednu od ponuđenih vrijednosti, željet ćete joj pristupiti. Bez obzira je li ju ručno upisao ili izabrao među ponuđenima, ona će biti smještena u svojstvo `Text`. Međutim, postoji i svojstvo `SelectedItem` koje će sadržavati izraz iz kućice samo ako je on izabran među ponuđenima, a ako ga je korisnik ručno upisao, to će svojstvo biti prazno (prazan *string*).

Želite li znati indeks odabrane vrijednosti, koristit ćete svojstvo `SelectedIndex`. Osim čitanja, u to svojstvo možete i pisati te na taj način u kodu promijeniti izbor. Primjerice, želite li podesiti izbor na prvi član kolekcije, napisat ćete sljedeći izraz:

```
comboBox1.SelectedIndex = 0;
```

Često puta nećete znati indeks člana kolekcije koji želite postaviti kao odabrani. U tom će vam slučaju pomoći metoda FindString. Evo kako odabrati prvog člana koji počinje znakovima "abc":

```
comboBox1.SelectedIndex = comboBox1.FindString("abc")
```

Kao što iz primjera možete zaključiti, metoda FindString vraća indeks prvog člana u kolekciji koji počinje navedenim nizom znakova.

Lista

Kontrole tipa ListBox u mnogočemu su slične padajućim listama zbog istog načina pohrane svojih članova. Sve što smo rekli vezano uz kolekciju Items, vrijedi i ovdje, samo što se članovi kolekcije na ekranu drugačije prezentiraju.

Svojstvo BorderStyle određuje tip ruba kojim će lista biti uokvirena. Dostupne su vrijednosti None (bez ruba), FixedSingle (rub širine jednog piksela) i Fixed3D (trodimenzionalan rub).

Ukoliko ima više članova kolekcije nego što je veličina kontrole, pojavit će se tzv. *scrollbarovi*. Ako želite da oni uvijek budu prisutni na ekranu, svojstvo ScrollAlwaysVisible postavite na *true*.



Slika 8-13:
Usporedba kontrole ListBox s
isključenim (gore) i uključe-
nim (dolje) svojstvom
MultiColumn

III. DIO: DIJELOVI .NET-A

Članovi kolekcije `Items` na ekranu se mogu prikazivati na dva načina – sa stupcima i bez njih. Kako to izgleda na ekranu možete vidjeti na slici 8-13. Svojstvo koje određuje način prikaza naziva se `MultiColumn` i može poprimiti vrijednost *true* odnosno *false*. Ukoliko je prikaz po stupcima uključen, ima smisla podesiti i vrijednost svojstva `ColumnWidth`. Njime određujemo širinu stupca, a ukoliko to ne učinimo (odnosno, vrijednost svojstva bude 0), širina svakog stupca bit će automatski podešena ovisno o dužini članova kolekcije.

Namjena kontrole `ListBox` može biti puko prikazivanje liste, no korisniku možemo omogućiti i odabir jednog ili više članova. Namjenu kontrole određujemo svojstvom `SelectionMode`, a vrijednosti koje mu možemo pridružiti su `None`, `One`, `MultiSimple` i `MultiExtended`. Prva vrijednost ne dozvoljava označavanje članova, druga pruža mogućnost označavanja samo jednog, dok ostale omogućavaju odabir više članova istovremeno.

U slučaju da odaberete vrijednost `One`, označenoj vrijednosti ćete moći pristupiti preko svojstava `SelectionIndex` i `SelectedItem`. Prvo svojstvo sadržavat će indeks odabranog člana u kolekciji, dok će `SelectedItem` sadržavati njegovu vrijednost.

Treba li vam u programu funkcionalnost odabira više članova liste odjednom, morate odabrati između vrijednosti `MultiSimple` i `MultiExtended`. Razlika među njima svodi se na način na koji korisnik odabire članove. U prvom načinu klik na neoznačeni član učinit će ga označenim, dok će klik na označeni rezultirati neoznačenim. Ipak, u korisničkim sučeljima češće se koristi drugi način, koji omogućava označavanje više članova tek uz korištenje tipki `Shift` i `Ctrl` ili potezanjem miša pritisnute tipke, baš kao što je to napravljeno u samim Windowsima.

Većem broju označenih članova ne možemo pristupiti na isti način. Za te slučajeve postoje svojstva `SelectionIndices` i `SelectionItems`. To su kolekcije koje sadrže sve označene članove, prva njihove indekse u originalnoj kolekciji, a druga njihove vrijednosti.



I jednom označenom članu možete pristupiti preko svojstava `SelectionIndices` i `SelectionItems`. U tom će slučaju te kolekcije imati samo jednog člana s indeksom 0.

Uzmimo za primjer da nam treba kôd koji će sve označene članove iz kontrole `listBox1` prebaciti u `comboBox1`. To možemo napraviti na dva načina. Prvo, korištenjem svojstva `SelectedItems`:

```
comboBox1.Items.Clear(); // praznimo comboBox1
for (int n = 0; n < listBox1.SelectedItems.Count; n++)
{
    object vrijednostOznacenog = listBox1.SelectedItems[n];
```


8. POGLAVLJE: WINDOWS FORMS

```
comboBox1.Items.Add(vrijednostOznacenog);
}
```

Varijablu `n` vrtimo u petlji *for* sve dok je ona manja od ukupnog broja članova u kolekciji `listBox1.SelectedItems`. Unutar petlje varijabli `vrijednostOznacenog` tipa *object* pridružujemo jednu od vrijednosti iz kolekcije, onu koja ima indeks `n`. U sljedećem redu tu vrijednost dodajemo u `comboBox1`.

Istu smo stvar mogli napisati puno kraće, korištenjem naredbe *foreach*:

```
comboBox1.Items.Clear();
foreach (object vrijednostOznacenog in listBox1.SelectedItems)
{
    comboBox1.Items.Add(vrijednostOznacenog);
}
```

Drugi, nešto složeniji način je onaj u kojem koristimo svojstvo `SelectedIndices`:

```
comboBox1.Items.Clear();
for (int n = 0; n < listBox1.SelectedIndices.Count; n++)
{
    int indeksOznacenog = listBox1.SelectedIndices[n];
    comboBox1.Items.Add(listBox1.Items[indeksOznacenog]);
}
```

U ovom načinu iz kolekcije `listBox1.SelectedIndices` dobivamo indekse označenih članova. Stoga unutar petlje koristimo varijablu `indeksOznacenog` tipa *int*. U redu dodavanja kao parametar metode `Add` nećemo navesti samo varijablu, nego ćemo pomoću nje iz originalne kolekcije `listBox1.Items` izvući označene članove. I ovaj se način može osjetno skratiti korištenjem naredbe *foreach*, no to vam ostavljamo za samostalan rad.

Svojstva `SelectionIndices` i `SelectionItems` moguće je samo čitati. Za definiranje označenih članova koristite metodu `SetSelected`.



Želite li iz kôda označiti odnosno odznačiti članove kolekcije `listBox1.Items`, dobro će vam doći metoda `SetSelected`. Počnimo jednostavnim primjerom – označimo prva tri člana kolekcije:

III. DIO: DIJELOVI .NET-A

```
listBox1.SetSelected(0, true);
listBox1.SetSelected(1, true);
listBox1.SetSelected(2, true);
```

Iz primjera vidimo da metoda `SetSelected` zahtijeva dva parametra – indeks člana i vrijednost tipa *boolean* koja kazuje hoće li taj član biti označen (*true*) ili neoznačen (*false*).

Isprobajmo to na mrvicu složenijem primjeru, u kojem je cilj odznačiti sve članove kolekcije:

```
for (int n = 0; n < listBox1.Items.Count; n++)
{
    listBox1.SetSelected(n, false);
}
```

Primjer: Tekstualni editor

Nakon što smo iskoristili Beskorisnu aplikaciju kako smo znali i mogli, red je da napravimo novi primjer, pomoću kojega ćemo upoznati neke nove kontrole, svojstva i događaje.

Slika 8-14:
Ovo nam je cilj –
primjer bi na
kraju trebao izgledati ovako



Kao što možete vidjeti na slici 8-14, ne radi se o sasvim jednostavnoj aplikaciji. Sučeljem dominira veliko tekstualno polje za upis (`TextBox`, ovaj puta višelinijski), a oko njega su se smjestili

izbornik (MainMenu), traka s alatima (ToolBar) i statusna traka (StatusBar). Otvorite novi projekt i dovucite spomenute kontrole. Traka s alatima će se automatski smjestiti na vrh, statusna traka na dno, a izbornik će zasada biti isključivo izvan forme.

Polje za unos teksta (višelinijsko)

Prvo ćemo krenuti s podešavanjem polja za unos. Kako ćemo ga koristiti za uređivanje tekstualnih datoteka, moramo ga pretvoriti u višelinijsko polje za upis. To radimo mijenjanjem vrijednosti svojstva `MultiLine` na `true`. Nakon toga tu kontrolu možete razvući tako da zauzima sav prostor između dviju traka. Osim toga, svojstvo `ScrollBars` valja postaviti na `Both` kako bismo uključili *scrollbar*-ove koji će nam kod dužih tekstova omogućiti da vidimo i onaj dio koji ne stane na ekran.

Svojstvo `WordWrap` sigurno već prepoznajete po imenu. Radi se o opciji koja odlučuje hoće li se tekst u polju lomiti na kraju retka (kao što, primjerice, čini Word) ili izlaziti nadesno, izvan vidljivog dijela kontrole (kao što je uobičajeno kod editiranja programskog kôda).

Kod višelinijških polja važno je uključiti (postaviti na `true`) svojstva `AcceptsReturn` i `AcceptsTab`. Naime, ako su ona isključena, korisnik neće moći koristiti tipke `Enter` i `Tab` unutar teksta, već će njihovo stiskanje rezultirati pritiskom glavnog gumba na formi (`Enter`) odnosno prelaskom na sljedeću kontrolu (`Tab`). Kako se te dvije tipke vrlo često koriste prilikom uređivanja teksta (za prelazak u novi red i uvlačenje), takvo ponašanje ne bi bilo praktično.

U primjeru smo izmijenili i font kojim će slova u polju za unos biti ispisana. Naime, kod tekstualnih smo editora navikli na korištenje *monospaced* fonta pa je red da i naš primjer koristi takav (`Courier New`; 9pt).

Kod korištenja višelinijškog polja za unos teksta cijeli njegov sadržaj, baš kao i kod jednolinijškog, bit će zapisan u svojstvu `Text`. Međutim, želite li pristupati svakoj pojedinoj liniji, koristit ćete svojstvo `Lines`. To svojstvo nije ništa drugo nego kolekcija *stringova* tako da sve rečeno o njoj vrijedi i ovdje.

Ovaj tip polja potencijalno može imati potrebu primiti veliku količinu podataka tako da nam inicijalna vrijednost `MaxLength` od 32767 znakova vjerojatno neće biti dovoljna. Stoga trebamo to svojstvo postaviti na najveću moguću vrijednost odnosno broj 2147483647 (naime, to svojstvo je tipa `Int32`). To će nam omogućavati editiranje tekstualnih datoteka veličine do 2 GB!

Izbornik

Radite li iole napredniju aplikaciju, radi lakšeg ćete snalaženja u sučelju koristiti izbornik. Dodavanje izbornika u aplikaciju izuzetno je jednostavno jer je sva njegova funkcionalnost sadržana u kontroli `MainMenu`.

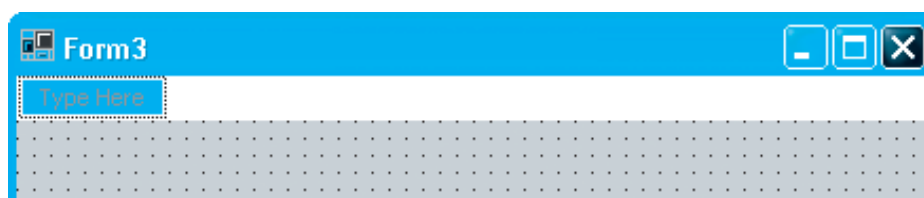
Nakon dodavanja kontrole na radnu površinu, valja krenuti s dodavanjem izbora. Nakon što je označite, na vrhu forme pojavit će se prazan izbornik s natpisom "Type Here" (slika 8-15) i ne ostaje vam drugo

III. DIO: DIJELOVI .NET-A

nego učiniti što kaže – počnete tipkati. Tako ćete napraviti svoju prvu stavku u izborniku, a nakon pritiska na tipku Enter, otvorit će se još mjesta s natpisom “Type Here” gdje možete dodavati nove stavke. Stavku po stavku i sagradit ćete izbornik poput ovoga na slici 8-16. Uočite da se tako unesena vrijednost zapisuje u svojstvo Text.

Slika 8-15:

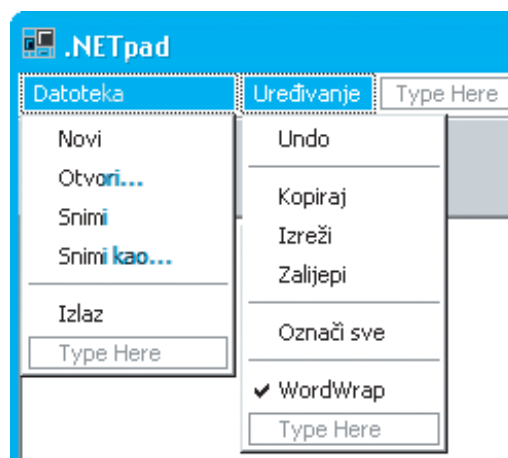
Netom dodana kontrola MainMenu



Uobičajeno je da se stavkama dodjeljuju tipkovničke kratice, kojima se može pristupiti kombinacijom Alt + slovo. Ako želite neko od slova pretvoriti u kraticu tog izbornika, prije njega, u svojstvu Text, stavite znak “&”. Primjerice, ako pridružimo vrijednost “Dat&oteka”, to će u aplikaciji biti napisano “Datoteka”, a tipkovnička kratica će biti Alt + O.

Slika 8-16:

Izbornik primjera Tekstualni editor u dizajnerskom načinu (ne pokušavajte ovo kod kuće, istovremeno otvaranje obje stavke je montirano)



8. POGLAVLJE: WINDOWS FORMS

Želite li u izborniku dodati liniju koja će odjeljivati dvije skupine opcija, kao tekst stavke upišite crticu (“-”).



Svaka stavka izbornika predstavlja zaseban objekt klase MenuItem, koji se automatski nazivaju po formuli menuItem1, menuItem2, menuItem3... Kako tako nazvani objekti u kasnijem pisanju kada mogu biti izuzetno nepraktični, odmah vam preporučujemo da ih preimenujete upisujući novu vrijednost u “svojstvo” Name. Mi smo ih u primjeru nazivali logično – stavka s tekstom Novi se zove menuNovi, Otvori je menuOtvori, Snimi je menuSnimi itd. Naravno, ime ne može sadržavati razmake, točke i neke druge specijalne znakove, što ovaj put ne uključuje tzv. hrvatske znakove; njihovo korištenje nije uobičajeno, pa smo, primjerice, stavku Uređivanje nazvali menuUredjivanje.

Primijetili ste da svaka stavka može imati podstavke. U tom odnosu, stavka koja ima podstavke naziva se roditelj (engl. parent), a podstavka se zove dijete (engl. child).



Svojstva stavaka

Sam izbornik nema svojstava koje biste željeli mijenjati, no stavke imaju. Ovdje treba naglasiti da svaka stavka predstavlja samostalan objekt pa ako želite svima promijeniti isto svojstvo, morate to napraviti za svaku posebno.

Želite li promijeniti neko svojstvo na više stavaka odjednom, prilikom označavanja istih držite tipku Ctrl. Ovo vrijedi samo za stavke unutar iste roditeljske stavke.

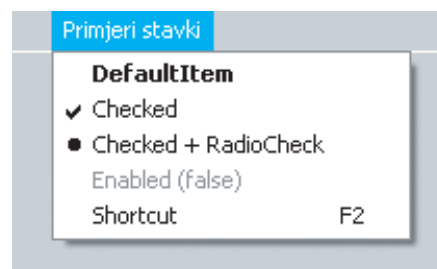


Kao i brojne druge objekte, svaku stavku možete sakriti mijenjajući svojstvo Visible. Ipak, u praksi se za tu svrhu češće koristi svojstvo Enabled, koje ne uklanja stavku iz izbornika, nego samo onemogućava njezino korištenje (stavka postaje “zasivljena”).

III. DIO: DIJELOVI .NET-A



Slika 8-17:
Stavke raznih tipova na jednom mjestu



Ukoliko stavka služi za uključivanje i isključivanje neke opcije ili mogućnosti, uključit ćete joj svojstvo `Checked`. Uključenost tog svojstva pokazuje kvačica na stavci, a kada je svojstvo isključeno, stavka izgleda kao i svaka druga. Ukoliko vam se kvačica ne sviđa, možete staviti kružić, uključivanjem svojstva `RadioCheck`. Ova dva svojstva su isključivo vizualne prirode – funkcionalnost uključivanja i isključivanja kvačice morat ćete napisati sami (kao što ćemo mi napraviti za neko liko stranica na našem primjeru).

Jednu od stavaka možete proglasiti *defaultnom* uključujući svojstvo `DefaultItem`. Takvoj će stavci ime biti ispisano podebljanim slovima, a bit će i pokrenuta, ako dvokliknete na njezinu roditeljsku stavku. Primjerice, da na izborniku poput ovoga na slici 8-17 dvokliknemo na roditeljsku stavku “Primjeri stavki” a da je prije ne otvorimo, rezultat bi bio isti kao da smo kliknuli na stavku “DefaultItem”. Ova se mogućnost vrlo rijetko koristi.



Svojstvo `DefaultItem` možete uključiti i za više stavaka pod istom glavnom stavkom. Tada će sve stavke biti podebljane, no samo prva će reagirati na dvoklik.

Jedno od najkorisnijih svojstava stavaka je mogućnost dodjeljivanja direktnih tipkovničkih kratice (tu ne mislimo na one u kombinaciji s tipkom `Alt`). To činimo mijenjajući svojstvo `Shortcut` (tip `System.Windows.Forms.Shortcut`). Ako je svojstvo `ShowShortcut` uključeno, onda će ta kratica biti prikazana uz stavku, no mnogo je važnija činjenica da se kontrola brine i za dostupnost ovako definirane tipkovničke kratice i kada je izbornik zatvoren. Drugim riječima, dodjeljivanjem tipkovničke kratice nekoj stavci automatski ste riješili postojanje kratice na nivou cijele forme (ali ne i aplikacije!).

Dodjeljivanje funkcionalnosti stavkama

I sami pogađate – klik na neku od stavaka stvara događaj `Click`, baš kao i klik na bilo koju drugu komponentu. Kao kod gumba, i ovdje je vrlo važno da te događaje povežete s funkcijama i u njima napišete neki kôd jer korisnici očekuju da svaka stavka nečemu služi.

8. POGLAVLJE: WINDOWS FORMS

Vratimo se na naš primjer i izbornik koji smo u njemu kreirali. Prvo ćemo programirati najjednostavnije stavke, a prva među njima je svakako stavka Izlaz. Dvokliknite na nju i dobivenoj funkciji dodajte sljedeći kôd:

```
private void menuIzlaz_Click(object sender, System.EventArgs e)
{
    Close();
}
```

Ostale stavke u ovoj skupini ostavljamo za kasnije, a sada ćemo za pozabaviti stavkama u skupini Uređivanje. Za prvih pet stavki kôd dajemo u kompletu, a koji pripada kojoj otkrit ćete prema imenima funkcija:

```
private void menuUndo_Click(...)
{
    textBox1.Undo();
}

private void menuKopiraj_Click(...)
{
    textBox1.Copy();
}

private void menuIzrezi_Click(...)
{
    textBox1.Cut();
}

private void menuZalijepi_Click(...)
{
    textBox1.Paste();
}

private void menuOznaciSve_Click(...)
{
    textBox1.SelectAll();
}
```

Jednostavno, zar ne? Dobro, moramo priznati, kao primjere smo koristili upravo one opcije za koje postoje gotove metode, dok smo lukavo izbjegli one koje zahtijevaju ikakvo dodatno programiranje. Međutim, niti takvi slučajevi nisu nuklearna fizika – često se stvari svode na tek nekoliko linija, kao što ćemo vrlo brzo vidjeti u primjerima snimanja i učitavanja datoteka.

III. DIO: DIJELOVI .NET-A

No, prije toga, pozabavimo se stavkom `WordWrap`, koja će imati ulogu uključivanja i isključivanja svojstva `WordWrap` koje pripada polju za unos teksta. Kod ovakvih stavaka vrlo je važno da prije svega uskladite početne vrijednosti svojstva `Checked` na stavci i svojstva `WordWrap` na polju za unos. Drugim riječima, ukoliko ste u dizajnerskom načinu svojstvo `WordWrap` ostavili uključeno, onda morate uključiti i svojstvo `Checked`.

Ovakvo ručno podešavanje nije problem u jednostavnim primjerima poput ovoga, no može stvoriti probleme u složenijim aplikacijama. Zašto onda ne bismo brigu o tome prepustili računalu? Napraviti ćemo da se prilikom pokretanja aplikacije provjerava stanje svojstva `WordWrap` i sukladno tome postavlja vrijednost svojstvu `Checked`. Kôd je sljedeći:

```
menuWordWrap.Checked = textBox1.WordWrap;
```

Ali, gdje ga staviti? Mogućnosti ima puno i sve će u našem primjeru bez problema raditi, no to ne mora vrijediti za neke druge složenije slučajeve. Jedno od rješenja je smjestiti kôd u konstruktor `Form1()`, na mjesto gdje je automatski generiran komentar "TODO: Add any constructor code after `InitializeComponent` call". Kako su konstruktori metode koje se izvršavaju prilikom stvaranja objekta, možemo biti sigurni da će se vrijednosti svojstava uskladiti na samom početku.

Osim tamo, kôd možete smjestiti i u funkciju vezanu uz događaj forme `Load` koji nastupa prilikom učitavanja forme, a ukoliko je moguće da neko svojstvo bude promijenjeno na više mjesta u programu, najsigurnije je raditi usklađivanje neposredno prije nego što se roditeljska stavka otvori. U tom slučaju kôd ćemo upisati u funkciju vezanu uz događaj `PopUp`, no ne na stavci na kojem ga mijenjamo nego na roditeljskoj stavci kojoj ona pripada. U našem slučaju, vezali bismo se na događaj `PopUp` objekta `menuUredjivanje`.



Pretpostavljamo da ste iz količine teksta koju smo posvetili usklađivanju shvatili koliko je ono važno. Takvi, naizgled sitni propusti, vrlo snažno utječu na korisnika zbunjujući ga i smanjujući povjerenje u program i autora. Jedna od glavnih snaga svih iole popularnijih aplikacija je doradenost sitnica.

Evo konačno i funkcije koju valja vezati uz klik na stavku `WordWrap`:

```
private void menuWordWrap_Click(...)
{
    // inverzija svojstva WordWrap:
    textBox1.WordWrap = !(textBox1.WordWrap);
}
```



```
// usklađivanje sa stavkom u izborniku:  
menuWordWrap.Checked = textBox1.WordWrap;  
// prethodna linija nije potrebna ako radimo  
// usklađivanje uz događaj PopUp  
}
```

Rad s datotekama

Kad u naprednim uređivačima teksta odaberete opciju otvaranja novog dokumenta, otvorit će vam se nov, prazan list papira u novom prozoru. U jednostavnijim editorima (kakav je naš), koji nemaju mogućnost otvaranja više dokumenata istovremeno, otvaranje novog dokumenta jednako je brisanju sadržaja postojećeg polja za upis. Stoga nam je funkciju otvaranja novog dokumenta vrlo jednostavno napisati:

```
private void menuNovi_Click(...)  
{  
    textBox1.Text = "";  
}
```

Dijaloški okvir za otvaranje datoteka

Nešto složenija funkcionalnost je učitavanje datoteke s diska i prikaz njezina sadržaja u polju za unos teksta. Taj ćemo posao podijeliti u dva dijela. Prvi nam je zadatak ponuditi korisniku popis datoteka na disku i mogućnost da jednu od njih izabere, dok će nam drugi biti otvaranje izabrane datoteke i kopiranje njezina sadržaja u textBox1.

Prvi zadatak, iako se na prvi pogled čini složen, zapravo je vrlo jednostavan. Naime, za tu ćemo funkcionalnost koristiti kontrolu OpenFileDialog koji nam sve potrebno donosi kao na pladnju. Dodajmo, stoga, kontrolu OpenFileDialog na formu (odnosno pokraj nje) i pozabavimo se nekim njezinim svojstvima.

OpenFileDialog zapravo je dijaloška forma koja na sebi ima sve potrebne kontrole koje omogućavaju korisniku da odabere datoteku. Prije nego što je prikažemo na ekranu, zgodno je podesiti svojstva koja utječu na njen izgled i ponašanje. Tako, primjerice, svojstvo Title određuje koji će se naslov naći u zaglavlju prozora.

Svojstvo InitialDirectory određuje mapu koja će biti prikazana prilikom otvaranja prozora. S ovim svojstvom treba postupati oprezno jer nema stopostotne garancije da mapa koju ovdje navedete postoji na računalu osobe koja koristi program. Zato u njega valja trpati vrijednosti isključivo na temelju prethodnog odabira mape, analize korisnikovog diska ili slično. U slučaju da, kao mi u

III. DIO: DIJELOVI .NET-A

primjeru, ne upišete nikakvu vrijednost, program će kao inicijalnu mapu otvoriti “My Documents” odnosno zadnju korištenu mapu.

Slika 8-18:
Dijaloška forma
OpenFileDialog



Svojstvo `RestoreDirectory` jedno je od onih koji vam može natjerati suze na oči ako na njega zaboravite. Naime, inicijalno je isključeno, a to znači da će svaki put kada otvorite datoteku pomoću dijaloškog okvira `OpenFileDialog` mapa u kojoj se datoteka nalazi postati radna mapa programa. Zato preporučujemo da vrijednost ovog svojstva postavite na `true` kako vam ne bi interferirao s radnom mapom.



Radna mapa aplikacije je, ako nije drugačije postavljeno, ona mapa u kojoj se nalazi i izvršna datoteka aplikacije. Radnu mapu možete mijenjati u prečici programa (desni klik na *shortcut*, Properties, polje “Start in”) ili pak u samom programu. Putanju radne mape u svakom trenutku možete pročitati iz izraza `Environment.CurrentDirectory`. Inače, radna mapa služi kao mjesto gdje aplikacija traži sve datoteke kojima nije navedena putanja. Drugim riječima, ukoliko u kodu na neku datoteku referencirate bez njezine putanje, podrazumijevat će se da se nalazi u radnoj mapi.

8. POGLAVLJE: WINDOWS FORMS

Vjerojatno najvažnije svojstvo ovog dijaloškog okvira je ono koje je nazvano Filter. Pomoću njega određujemo sadržaj padajuće liste "Files of type" (vidi dno slike 8-18) odnosno omogućavamo filtriranje prikazanih datoteka prema njihovoj ekstenziji. To svojstvo, u usporedbi s drugima, ima vrlo neobičnu sintaksu, a kod pisanja iste morat ćete se osloniti isključivo na sebe jer ne postoji nikakav pomoćni prozor.

Radi se po posebnim pravilima napisanom *stringu*. Primjerice, želite li omogućiti korisnicima učitavanje tekstualnih datoteka (dakle, ekstenzija će biti "txt"), svojstvo Filter glasit će ovako:

```
Text files (*.txt)|*.txt
```

U prvi dio možete upisati što vam je drago (iako je način iz primjera uvriježen i ne preporučujemo izmišljanje tople vode), dok u drugom dijelu, nakon znaka "|", upisujete sam filter (zvjezdica zamjenjuje neodređen broj bilo kojih znakova).

Ako želite da padajuća lista "Files of type" ima više članova, napisat ćete nešto poput ovoga, po formuli opis, crta, filter, crta, opis, crta, filter...

```
Text files (*.txt)|*.txt|All files (*.*)|*.*
```

Jednom članu padajuće liste možete dodijeliti i više ekstenzija. To onda izgleda ovako:

```
Images (*.jpg; *.gif; *.png)|*.jpg;*.gif;*.png
```

Svojstvom `FilterIndex` određujete koji će od članova padajuće liste inicijalno biti odabran (obavezno obratite pažnju na upozorenje uz tekst!).

Za razliku od drugih indeksa, prvi član u svojstvu `FilterIndex` određuje se brojkom 1 (a ne 0, kako smo navikli).



Svojstva `CheckFileExists` i `CheckPathExists` gotovo sigurno nećete mijenjati kod dijaloškog okvira za otvaranje datoteka jer je sasvim logično da datoteku možete otvoriti jedino ako postoji, a ova svojstva, ako su uključena, provjeravaju upravo to. Još manje utjecaja imaju svojstva `AddExtension` i `DefaultExt` koja ćemo obraditi malo kasnije, kod dijaloškog okvira za snimanje.

U svojstvo `FileName` zapisuje se ime i putanja do datoteke koju je korisnik odabrao i ona se u pravilu ne koristi prije, nego nakon pozivanja dijaloškog okvira.

III. DIO: DIJELOVI .NET-A

U slučaju da želite otvoriti više datoteka odjednom (što u našem primjeru nema smisla, no može vam zatrebati), prije poziva okvira svojstvu `MultiSelect` dodijelite vrijednost `true`. Imajte na umu da ćete u tom slučaju sami iz parametra `FileName` morati razlučiti putanje svih odabranih datoteka.

Pretpostavljamo da svojstva podešavate kroz pomoćni prozor sučelja Visual Studija, no pozivanje dijaloškog okvira možete napraviti samo iz kôda, najčešće vezano uz neki događaj. Kod nas će to biti funkcija vezana za odabir stavke `Otvori` u izborniku. Pozivanje okvira radimo metodom `ShowDialog()`, a ona se gotovo uvijek koristi u ovakvom obliku:

```
private void menuOtvori_Click(...)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        // kôd za čitanje datoteke
    }
}
```

Koncentrirajmo se za trenutak na liniju koja počinje naredbom *if*. U uvjetu koji nakon naredbe koristimo primjećujete kako metodu za pozivanje dijaloškog okvira uspoređujemo s vrijednošću `DialogResult.OK`. Naime, ta metoda vraća vrijednost tipa `DialogResult`. Ukoliko je korisnik u okviru izabrao datoteku i pritisnuo gumb `OK`, onda će vraćena vrijednost biti jednaka `DialogResult.OK`. Ukoliko je odustao pritisnuvši gumb `Cancel`, onda će vrijednost biti `DialogResult.Cancel`. Slijedi sasvim logičan zaključak – kôd koji će odraditi otvaranje datoteke treba biti izvršen samo ako je korisnik kliknuo `OK`. Zato vraćenu vrijednost metode `ShowDialog()` uspoređujemo s adekvatnom vrijednošću tipa `DialogResult` i samo ako su identične dozvoljavamo izvršavanje kôda za otvaranje datoteke.



Spomenuti način provjere, kao što ćemo vidjeti, jednak je za sve dijaloške okvire, s tim da neki, primjerice, vraćaju vrijednosti `DialogResult.Yes` i `DialogResult.No`.

Čitanje datoteke s diska

Dopunimo prošli komad kôda i dodajmo unutar bloka pod paskom naredbe *if* četiri linije za otvaranje datoteke i čitanje njezina sadržaja:

```
private void menuOtvori_Click(...)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        // kôd za čitanje datoteke
    }
}
```

8. POGLAVLJE: WINDOWS FORMS

```

{
    string ImeDatoteke = openFileDialog1.FileName;
    System.IO.StreamReader Datoteka
        = new System.IO.StreamReader(ImeDatoteke);
    textBox1.Text = Datoteka.ReadToEnd();
    Datoteka.Close();
}

```

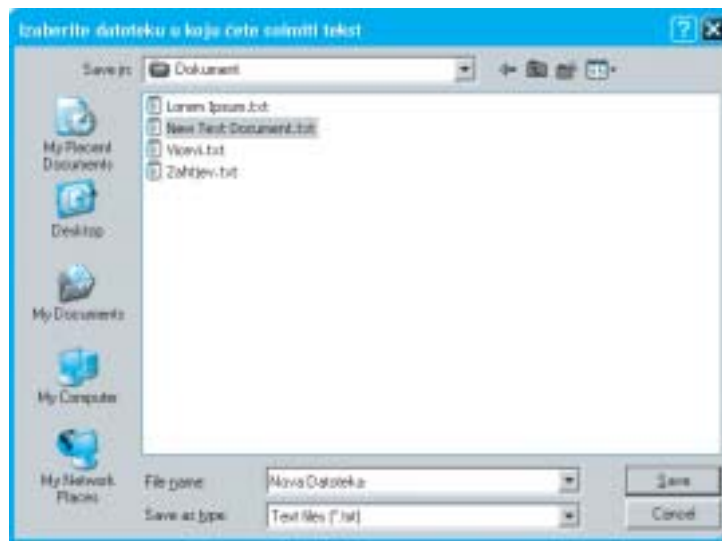
U prvoj liniji deklariramo varijablu *ImeDatoteke* tipa *string*, kojoj odmah pridružujemo ime i putanju datoteke koja je izabrana pomoću dijaloškog okvira (svojstvo *FileName*).

Sljedeći redak stvara objekt nazvan *Datoteka*. To je objekt tipa *StreamReader* (u klasi *System.IO*) koji, između ostalog, omogućava čitanje datoteka s diska. Kao parametar konstruktora navodimo varijablu *ImeDatoteke* koja sadrži ime i putanju do datoteke. Svaki put kada pristupamo tom objektu zapravo radimo s datotekom koju je korisnik odabrao.

Treća linija bloka u svojstvo *textBox1.Text* sprema sadržaj datoteke koji uzimamo metodom *ReadToEnd()*. U zadnjoj liniji zatvaramo datoteku jer je više nećemo koristiti.

Dijaloški okvir za snimanje datoteka

Za snimanje podatka u datoteku predvidjeli smo dvije stavke: *Snimi (Save)* i *Snimi kao (Save as)*. Dijaloški okvir za snimanje datoteka trebat će nam prvenstveno u drugoj, iako se potreba za njime može pojaviti i u prvoj ukoliko datoteka još nije bila snimana.



Slika 8-19:
Dijaloška forma
SaveFileDialog

III. DIO: DIJELOVI .NET-A

Funkcija vezana uz događaj odabira stavke Snimi kao izgledat će ovako:

```
private void menuSnimiKao_Click(...)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        // kôd za snimanje datoteke
    }
}
```

Uočite da je pozivanje ovog dijaloškog okvira identično onome za otvaranje datoteka, čak se koristi i isti tip vraćene vrijednosti. Osim toga, ove dvije kontrole dijele i neka zajednička svojstva tako da dio priče o svojstvima kontrole OpenFileDialog vrijedi i ovdje.

Međutim, dio tih svojstava u ovom slučaju ima više smisla. Tu prvenstveno mislimo na svojstva CheckPathExists i CheckFileExists. Kod snimanja je, kao i kod otvaranja, bitno da putanja (CheckPathExists) postoji jer u protivnom, ako korisnik ručno upiše ime mape koja ne postoji, datoteka u njoj neće moći biti kreirana (u slučaju da to dozvolite, prije snimanja ćete u kodu morati kreirati tu mapu). Međutim, svojstvo CheckFileExists bi trebalo biti na vrijednosti *false* jer kod snimanja datoteke ne samo da nije nužno da datoteka već postoji nego je i poželjno da ne bude tako. To ne znači da korisnik neće moći prepisati neku postojeću datoteku, no to je već domena svojstva OverwritePrompt koji, ako je uključen, upozorava korisnika da će prepisati postojeću datoteku ukoliko upiše ime datoteke koje već postoji.

Ako iz nekog razloga želite da korisnik bude upozoren i ako kreira sasvim novu datoteku, uključit ćete svojstvo CreatePrompt. Ova se mogućnost koristi vrlo rijetko.

Svojstva AddExtension i DefaultExt također ovdje imaju puno više smisla, iako smo ih susretali i kod prošlog dijaloškog tipa. Naime, korisnici su navikli ne pisati ekstenzije svojim datotekama, već očekuju da se o tome brine sustav. Primjerice, u programima za editiranje tekstualnih datoteka korisnik logično je da će se automatski dodati ekstenzija “txt”, pa smo mi u svom primjeru svojstvo AddExtension postavili na *true*, a u svojstvo DefaultExt zapisali ekstenziju koju želimo da bude dodana, ako je korisnik sam ne upiše (dakle, upisali smo vrijednost “txt”). Kao rezultat toga, kada korisnik upiše samo “datoteka”, dijaloški okvir će automatski dodati ekstenziju i u svojstvu FileName vratiti vrijednost “c:\neka putanja\datoteka.txt”.

Snimanje datoteka na disk

Sljedeći zadatak nam je napisati kôd koji će snimati sadržaj tekstualnog polja u datoteku. Kako ćemo tu funkcionalnost koristiti na dva mjesta (kod stavaka Snimi i Snimi kao), preporučljivo je da napravimo zasebnu funkciju koja će imati tu funkcionalnost te je kasnije pozivamo s mjesta na kojem nam treba:

8. POGLAVLJE: WINDOWS FORMS

```
private void snimanjeDatoteke(string ImeDatoteke)
{
    System.IO.StreamWriter Datoteka
        = new System.IO.StreamWriter(ImeDatoteke, false);
    Datoteka.Write(textBox1.Text);
    Datoteka.Close();
}
```

Kao parametar ove funkcije tražimo ime datoteke i zapisujemo u varijablu `ImeDatoteke` te je kasnije u kodu koristimo. Primijetiti ćete da u ovom primjeru koristimo drugi tip objekta (`System.IO.StreamWriter`) koji služi za pisanje u datoteke. Prilikom kreiranja objekta opet navodimo ime datoteke, a kao drugi parametar pojavljuje se opcija dodavanja ili prepisivanja datoteke. Konkretno, ako kao drugi parametar navedemo vrijednost *false*, stara će datoteka biti prepisana, a ako stavimo vrijednost *true*, onda će ono što pišemo biti dodano na kraj postojeće datoteke. Naravno, sve pod uvjetom da stara datoteka istog imena postoji – ukoliko ne postoji, parametar nema utjecaja na ponašanje objekta.

Ostale dvije linije sigurno i sami razumijete. Prva metodom `Write()` u datoteku zapisuje sadržaj svojstva `textBox1.Text`, a druga zatvara datoteku.

Još nam preostaje napisati kôd za stavke `Snimi` i `Snimi kao`. Počet ćemo s potonjom:

```
private void menuSnimiKao_Click(...)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        snimanjeDatoteke(saveFileDialog1.FileName);
    }
}
```

Kako već imamo pripremljenu funkciju za snimanje datoteke, pozivamo je, a kao parametar joj navodimo ime i putanju datoteke koju je korisnik izabrao pomoću dijaloškog okvira za snimanje.

Stavka `Snimi` nešto je složenija. Razmislimo malo... Ukoliko dokument još nije bio sniman, onda se stavka `Snimi` treba ponašati isto kao i stavka `Snimi kao` – otvoriti dijaloški okvir i pitati za ime. Međutim, ukoliko je dokument bio sniman ili smo otvorili postojeću datoteku, onda znamo njezino ime i nema potrebe da program otvara dijaloški okvir za biranje imena.

Stoga nam treba mjesto gdje ćemo zapisati ime datoteke kada je otvorimo i/ili snimimo. Slobodni smo za to stvoriti na nivou klase posebnu varijablu tipa *string*, no puno zgodnije rješenje je iskorištavanje jednog dosad nespomenutog svojstva, prisutnog kod apsolutno svih kontrola. Svojstvo se zove `Tag` i pripada tipu *object*, što znači da u njega možemo utrpiti bilo što, uključujući i ime datoteke. Nadogradimo malo naše funkcije kako bi ime datoteke zapisali u svojstvo `Tag` kontrole `textBox1`:

III. DIO: DIJELOVI .NET-A

```
private void menuNovi_Click(...)
{
    textBox1.Text = "";
    textBox1.Tag = "";
}

private void menuOtvori_Click(...)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        string ImeDatoteke = openFileDialog1.FileName;
        System.IO.StreamReader Datoteka
            = new System.IO.StreamReader(ImeDatoteke);
        textBox1.Text = Datoteka.ReadToEnd();
        Datoteka.Close();
        textBox1.Tag = ImeDatoteke;
    }
}

private void snimanjeDatoteke(string ImeDatoteke)
{
    System.IO.StreamWriter Datoteka
        = new System.IO.StreamWriter(ImeDatoteke, false);
    Datoteka.Write(textBox1.Text);
    Datoteka.Close();
    textBox1.Tag = ImeDatoteke;
}
```

U slučaju stvaranja novog dokumenta on neće automatski imati pripadajuće ime datoteke, pa svojstvu `textBox1.Tag` pridružujemo prazan *string*. Kod učitavanja datoteke, u to svojstvo zapisujemo ime datoteke koje je korisnik izabrao u dijaloškom okviru, a kod snimanja koristimo parametar funkcije `snimanjeDatoteke` u kojem se ime datoteke nalazi. Promjena imena datoteke vezane uz dokument koji editiramo može se, dakle, dogoditi u tri slučaja – prilikom stvaranja novog dokumenta, otvaranja druge datoteke s diska i snimanja datoteke.

Nakon što smo se uvjerali da se u svojstvu `textBox1.Tag` uvijek nalazi pravo ime datoteke i da će to svojstvo, ukoliko ime ne postoji, biti prazno, možemo napisati funkciju za stavku `Snimi`:

```
private void menuSnimi_Click(object sender, System.EventArgs e)
{
    if (textBox1.Tag.ToString() != "")
```



```

    {
        snimanjeDatoteke(textBox1.Tag.ToString());
    }
    else
    {
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
            snimanjeDatoteke(saveFileDialog1.FileName);
        }
    }
}

```

Kako je `textBox1.Tag` tipa *object*, bitno je pomoću metode `ToString()` napraviti konverziju u *string*, kako bismo to svojstvo mogli koristiti u uvjetu. Ukoliko ime datoteke postoji (svojstvo `Tag` nije prazan *string*), poziva se funkcija za snimanje datoteke, a kao parametar navodimo ime datoteke odnosno isto to svojstvo `Tag`. U suprotnom slučaju, izvršit će se izraz identičan onome u funkciji vezanoj uz stavku `Snimi kao`. Umjesto te tri linije, mogli smo napisati i sljedeće:

```

menuSnimiKao.PerformClick();

```

Metoda `PerformClick()`, naime, simulira klik na objektu kojem pripada, nastupa događaj `Click` te se izvršava funkcija vezana uz njega (u našem slučaju, funkcija `menuSnimiKao_Click`).

Snimanje umjesto gubitka teksta

Ukoliko radite po primjerima i isprobavate sve što radimo (a to toplo preporučujemo), primijetit ćete da nam nedostaje još jedna sitnica kako bi se naša aplikacija ponašala u skladu s običajima. Naime, naučili smo da prilikom stvaranja novog dokumenta ili otvaranja nekog drugog, program upozorava da postojeći dokument nije snimljen i ponudi njegovo snimanje. U našem primjeru takvog ponašanja nema pa ako, primjerice, slučajno kliknete na novi dokument, a postojeći niste snimili, sve će vam promjene otići u nepovrat.

Zato nam treba pokazatelj koji će u svakom trenutku znati je li tekst u polju za upis promijenjen od zadnjeg učitavanja, snimanja odnosno stvaranja novog dokumenta.

Da bismo riješili taj problem, za početak valja definirati varijablu tipa *boolean* na nivou cijele klase. To radimo tako da na vrhu kôda, u predjelu za deklaraciju varijabli, iznad konstruktora klase `public Form1()`, upišemo sljedeće:

```

private bool Mijenjano = false;

```

Na taj način početnu vrijednost varijable `Mijenjano` stavljamo na *false*. U slučaju da se tekst u kontroli `textBox1` izmijeni, varijablu bi trebalo postaviti na *true*. Zato sljedeću funkciju vežemo uz događaj `TextChanged`:

III. DIO: DIJELOVI .NET-A

```
private void textBox1_TextChanged(...)
{
    Mijenjano = true;
}
```

Zahvaljujući ovom kodu, vrijednost varijable `Mijenjano` bit će gotovo uvijek *true*. Ipak, ne zaboravimo da ova varijabla označava je li tekst mijenjan od zadnjeg snimanja. Sukladno tome, neposredno nakon snimanja varijabla `Mijenjano` bi trebala imati vrijednost *false*. Ista stvar je i u situacijama neposredno nakon otvaranja dokumenta (tekst je identičan onome u datoteci, što znači da nije mijenjan) odnosno nakon stvaranja novog dokumenta (dokument je prazan, pa se nema što snimati). Zato u sve te funkcije (a to su, igrom slučaja, sve one u koje smo dodavali liniju za pamćenje naziva datoteke) dodajemo sljedeću liniju kôda:

```
Mijenjano = false;
```

Pokazatelja je li tekst promijenjen imamo, pa nam preostaje samo njegovo korištenje. Zato ćemo napisati jednostavnu funkciju:

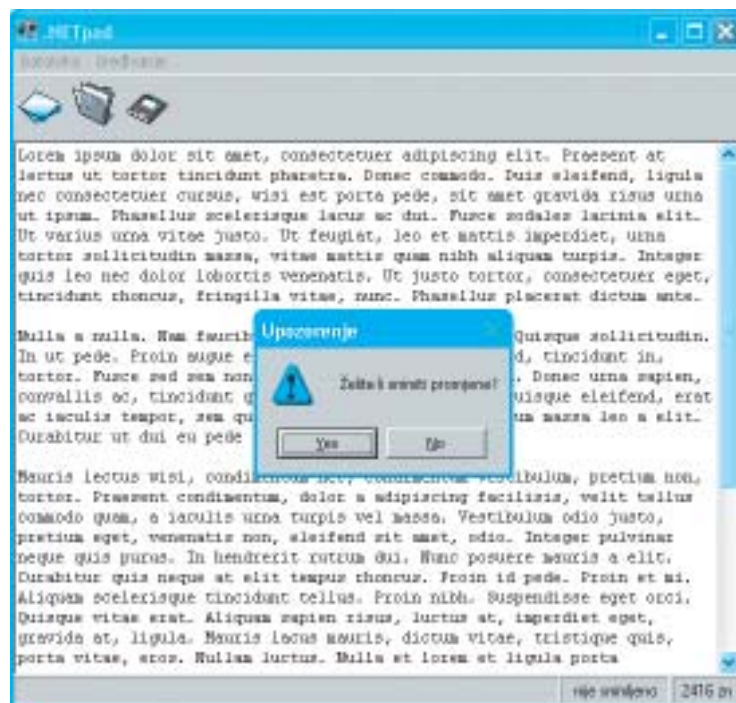
```
private void provjeraSnimljenosti()
{
    if (Mijenjano)
    {
        if (MessageBox.Show(
            "Želite li snimiti promjene?",
            "Upozorenje",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Exclamation) == DialogResult.Yes)
        {
            menuSnimi.PerformClick();
        }
    }
}
```

Osnovna zadaća ove funkcije je da, ukoliko je tekst mijenjan, pozove dijaloški okvir s pitanjem "Želite li snimiti promjene?". Ako korisnik odgovori potvrdno, pokrenut će se procedura snimanja simulacijom klika na izborničku stavku `Snimi`. Ona je u ovom slučaju idealna – ukoliko postoji ime dokumenta, snimit će promjene pod tim imenom, a ukoliko ne postoji, otvorit će dijaloški okvir za upis imena.

Dio u primjeru koji još nismo sreli odnosi se na dijaloški okvir `MessageBox`. Iako se na prvi pogled čini slična dijaloškim okvirima koje smo upoznali, ona ima specifičnu karakteristiku da nije po-

8. POGLAVLJE: WINDOWS FORMS

trebno (zapravo, nije moguće) napraviti novu instancu te klase odvlačenjem pripadajuće kontrole na formu. Ona se koristi bez ikakvih predradnji, jednostavnim pozivanjem statične (zajedničke; vidi prethodno poglavlje) metode Show() kojom kao parametre navodimo svojstva koja želimo da ima dijaloški okvir.



Slika 8-20:
Želite li snimiti promjene, pita MessageBox.

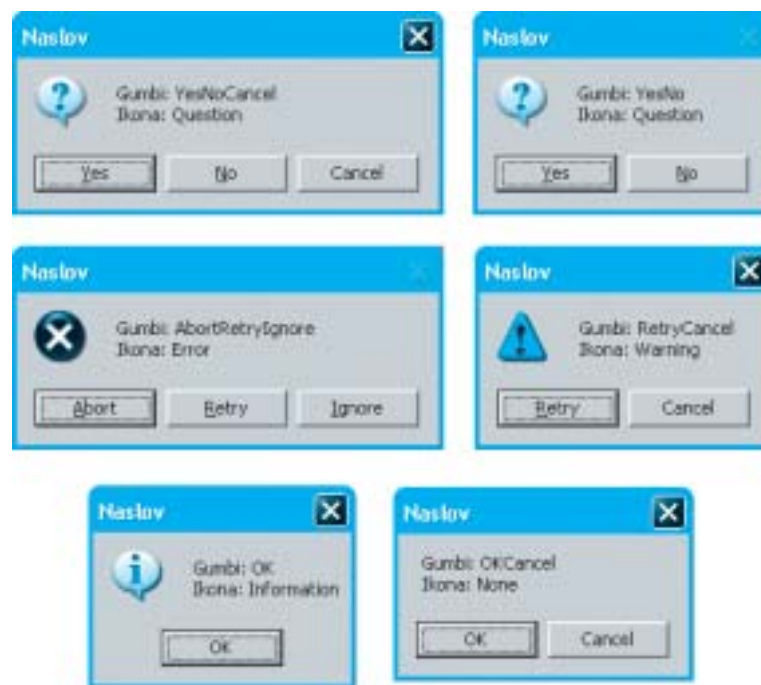
Metoda Show() je preopterećena što znači da može imati različite “komplete” parametara, a mi smo u primjeru izabrali jedan, u ovom slučaju najpraktičniji među njima. On ima četiri parametara – prvi predstavlja tekst (u našem slučaju pitanje) koji će se pojaviti, drugi služi za definiranje naslova u zaglavlju, treći određuje koji će gumbi biti dostupni (u našem slučaju Yes i No), a četvrti ikonu koja će biti prikazana (mi smo odabrali uskličnik). Na slici 8-21 možete vidjeti razne kombinacije spomenutih parametara.

Želite li u tekstu MessageBox-a prijeći u novi redak, koristite izraz “\n” (bez navodnika).



III. DIO: DIJELOVI .NET-A

Slika 8-21:
Gumbi i ikone dostupni u MessageBox-u (neke od dostupnih ikona nisu prikazane, jer predstavljaju stari način imenovanja)



Koji je gumb korisnik pritisnuo otkriva se na isti način kao i kod ostalih dijaloških okvira. Naravno, valja paziti na povezanost vrijednosti koje mogu biti vraćene i prikazanih gumbi, posebno stoga što vas kompajler o propustima tog tipa neće obavijestiti.

Konačno, napisanu funkciju treba pozvati prije svih radnji koje mogu rezultirati gubitkom podataka u kontroli textBox1. Tri su takve radnje – stvaranje novog dokumenta, otvaranje drugog dokumenta i izlaz iz programa. Stoga poziv te funkcije treba smjestiti na sam početak tih funkcija. Evo kako će to izgledati:

```
private void menuNovi_Click(...)
{
    provjeraSnimljenosti();
    textBox1.Text = "";
    textBox1.Tag = "";
}

private void menuOtvori_Click(object sender, System.EventArgs e)
{
    provjeraSnimljenosti();
```

```

    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        // dio kôda je izostavljen
    }
}

```

Kod izlaza iz programa ovaj način ne “pali” jer je, osim klikom na stavku Izlaz, iz programa moguće izaći klikom na crveni križić u gornjem desnom uglu ili kombinacijom tipaka Alt + F4. Zato tu funkciju treba vezati uz događaj forme, i to onaj koji nastupa neposredno prije zatvaranja – Closing. Sâm kôd je vrlo jednostavan:

```

private void Form1_Closing(...)
{
    provjeraSnimljenosti();
}

```

Traka s alatima

Kontrola ToolBox služi za kreiranje trake s alatima. Takav izbor najčešće korištenih funkcija možemo susresti u gotovo svim aplikacijama.

Prvi korak je postavljanje kontrole na formu, nakon čega će se ona automatski prilijepiti uz gornji obod forme. Nakon toga joj treba definirati pravila izgleda i ponašanja te naposljetku dodati određeni broj gumbi i odrediti im funkcionalnosti. Krenimo redom...

Među vizualnim svojstvima trake s alatima pronaći ćemo brojne već poznate stavke, ali i neke nove, specifične za ovu kontrolu. Od poznatih spominjemo tek BorderStyle koji određuje tip obruba, dok od novih valja spomenuti svojstvo Divider koji određuje postojanje linije koja razdvaja sam vrh forme (odnosno izbornik, ako postoji) s alatnom trakom. Svojstvo Wrappable određuje hoće li gumbi na traci moći biti prikazani i u više redova ukoliko veličina prozora to zahtijeva.

Gumbi na traci

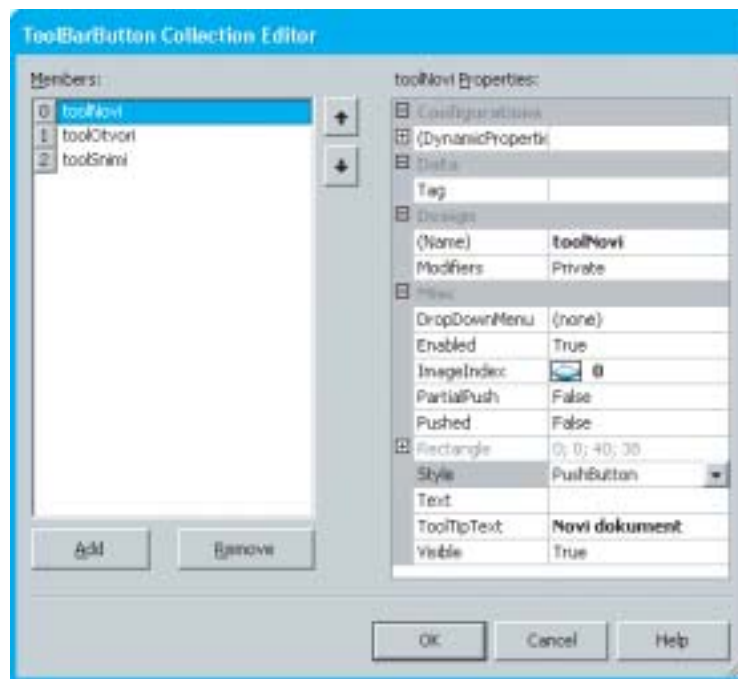
Dodavanje gumba na alatnu traku rješava se preko svojstva Buttons. Klik na gumbić kraj tog svojstva otvorit će pomoćni prozor (slika 8-22) unutar kojeg ćemo dodavati gumbе i dodjeljivati im svojstva.

Gumbi na alatnoj traci također su kontrole (tipa `ToolBarButton`), no njihovo dodavanje u Visual Studijevom sučelju riješeno je na drugačiji način nego što je to učinjeno sa stavkama izbornika koje su u vrlo sličnom odnosu sa svojom roditeljskom kontrolom (`MainMenu`). Ipak, na ovo ne možemo uzeti dokaz nedosljednosti jer je izbornik značajno složeniji od alatne trake pa zahtijeva i drugačije rješenje. Naime, svaka (pod)stavka može imati vlastite podstavke, dok gumbi alatne trake mogu biti isključivo u istom nivou.



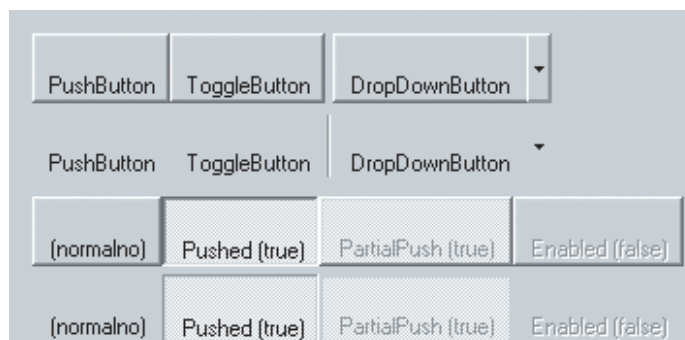
III. DIO: DIJELOVI .NET-A

Slika 8-22:
Pomoćni prozor za dodavanje gumba



Kako je svaki gumb zapravo kontrola (vidi okvir), svaki od njih će imati zasebna svojstva koja možemo prilagođavati na desnoj strani pomoćnog prozora. Ovdje vrijedi isti savjet kao i za stavke izbornika – imenujte ih logično kako biste se kasnije lakše snalazili.

Slika 8-23:
Različiti tipovi, stilovi i pojave gumba



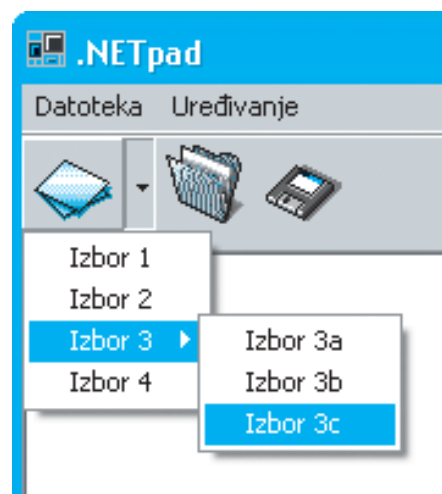
8. POGLAVLJE: WINDOWS FORMS

Gumbi se na alatnoj traci mogu pojavljivati u dva oblika – kao klasični izbočeni gumbi ili moderniji ravni gumbi. To odlučujemo na nivou cijele alatne trake mijenjajući njeno svojstvo Appearance. Drugim riječima, nije moguće na istoj traci imati i jedan i drugi tip gumba.

Kako će se gumb ponašati određujemo za svaki posebno (dakle, u pomoćnom prozoru iza svojstva Buttons). Tamo možemo odabrati stil gumba koji može biti klasičan (PushButton), sklopni (ToggleButton), odjelni (Separator) i s padajućim izbornikom (DropDownButton). Kako koji stil izgleda u praksi možete vidjeti na slici 8-23 (prvi red su izbočeni, a drugi ravni).

Na toj slici postoje dvije stvari koje vas mogu zbuniti. Prvo je razlika između stila PushButton i ToggleButton – vizualno, među njima nema razlike, no pokrenete li aplikaciju, uočit ćete da se drugačije ponašaju. Naime, kad kliknete na gumb sa stilom PushButton, on će se nakon klika vratiti u početni položaj, dok će onaj sa stilom ToggleButton ostati pritisnut sve dok ga ponovo ne pritisnete. Osim vizualnog ponašanja, gumbima različitih stilova ćete drugačije i postupati u kodu – PushButton će izvršavati određenu radnju dok će ToggleButton uključivati odnosno isključivati određenu opciju, baš kao da se radi o stavkama u izborniku među kojima jedna ima, a druga nema kvačicu.

Druga zbunjujuća okolnost vezana uz sliku 8-23 jest gumb stila Separator. Naime, to zapravo i nije pravi gumb već linija koja odvaja dvije skupine gumba, opet vizualno na isti način kao što smo to radili u izbornicima. Međutim, tu već možemo pričati o nedosljednosti – dok u izbornicima separator dodajemo upisujući znak minusa umjesto teksta, ovdje to radimo pomoću svojstva stila.



Slika 8-24:
Gumb s padajućim izbornikom u akciji

Kod gumba stila ToggleButton postoji svojstvo preko kojeg možemo saznati je li gumb u pritisnutom stanju. Radi se o svojstvu Pushed koje poprima *boolean* vrijednosti. Osim njega, tu je i svojstvo

III. DIO: DIJELOVI .NET-A

PartialPush koje, ako je uključeno, na neobičan način zasivljuje gumb iako su sve njegove funkcije normalne. Kako to izgleda u praksi, možete vidjeti u zadnja dva retka slike 8-23, a osim spomenutih svojstva prikazali smo i svojstvo Enabled. Ono je moguće uključiti gumbima svih stilova, a osim što ih zasivljuje, ujedno i onemogućava njihovo korištenje.

Stil DropDownButton omogućava da gumbu dodamo i padajući izbornik. Padajući izbornik koji će se pojaviti, pogađate, posebna je kontrola. Radi se o kontroli tipa ContextMenu, koju je potrebno zasebno dovući na formu i povezati je s gumbom preko svojstva DropDownMenu.



U padajućoj listi svojstva DropDownMenu pronaći ćete i izbornik tipa MainMenu, kao sve izborničke stavke koje postoje na formi, no jedini dozvoljeni izbor je kontrola tipa ContextMenu.

ContextMenu je vrlo sličan kontroli MainMenu, s tom razlikom da se sve stavke moraju smjestiti ispod jedne, glavne “stavke” nazvane ContextMenu. Ta će se glavna “stavka” prikazati samo u dizajnerskom načinu rada.



Kontrolu ContextMenu možete vezati uz velik broj kontrola. Primjerice, ako je vežete uz formu preko svojstva ContextMenu, onda će taj izbornik biti prikazan kada korisnik na formu klikne desnom tipkom miša. Isto vrijedi i za brojne druge kontrole.

Gumbi stila DropDownButton mogu svoj padajući izbornik pokazati na dva načina. Prvi, uobičajeniji, kakav možete vidjeti na slici 8-24, uključuje postojanje strelice pritiskom na koju se izbornik otvara. U tom slučaju gumb može imati jednu funkcionalnost, a kroz stavke padajućeg izbornika mogu se ponuditi dodatne, u praksi srodne funkcionalnosti. Ako isključimo postojanje strelice (svojstvo DropDownArrows, na nivou cijele alatne trake), padajući izbornik će se otvoriti klikom na bilo koji dio gumba i on neće moći imati vlastitu funkcionalnost prilikom tog klika (drugim riječima, klikom na takav gumb neće nastupiti događaj ButtonClick, koji ćemo uskoro upoznati).

8. POGLAVLJE: WINDOWS FORMS

U Visual Studiju .NET 2003 postoji greška vezana uz svojstvo `DropDownButton`. Naime, dokumentacija govori da je defaultna vrijednost *false* i to funkcionira u dizajnerskom načinu, no kad pokrenete aplikaciju primijetit ćete da je svojstvo uključeno. Kada želite da to bude tako, vrijednost svojstva stavite na *true* i stvar će funkcionirati, no ako želite isključiti postojanje strelice, morat ćete se poslužiti trikom. U konstruktor forme na kojoj se alatna traka nalazi ubacite sljedeći kôd:

```
toolBar1.DropDownArrows = false;
```



Svaki gumb u alatnoj traci može imati tri karakteristike koje će korisniku govoriti o njegovoj funkcionalnosti. Tekst, sliku (ili, prikladnije rečeno, ikonu) i tzv. *tooltip*, mali žuti pravokutnik koji se pojavljuje s objašnjenjem nakon što nekoliko trenutaka zadržite miša iznad gumba (ili neke druge kontrole).

Tekst svakog gumba upisujemo u svojstvo `Text`, a na nivou cijele alatne trake svojstvom `TextAlign` možemo podesiti kako će taj tekst biti prikazan – ispod sličice (*Underneath*) ili desno od nje (*Right*). Naime, za gumbe u alatnoj traci očekuje se da imaju sličice; čak i u slučajevima kad ih ne definirate, za njih će biti “rezerviran” prostor na gumbu.

Sličice (ikone) postavljamo na već viđeni način, pomoću komponente `ImageList`. U nju spremimo sve sličice koje će nam trebati, povežemo je s kontrolom `ToolBar` preko svojstva `ImageList` i onda u svakom gumbu definiramo indeks sličice koji će on koristiti, u svojstvu `ImageIndex`.

Tooltipove definiramo također za svaki gumb zasebno, upisujući ga u svojstvo `ToolTipText`, dok na razini cijele trake s alatima određujemo hoće li se oni i prikazivati, pomoću svojstva `ShowToolTips`.

Veličina alatne trake određuje se automatski, ako je svojstvo `AutoSize` uključeno. Ako nije, onda se veličina neće prilagođavati veličini gumba. Želite li pak utjecati na veličinu gumba, možete kroz svojstvo `ButtonSize` podesiti veću vrijednost od one automatski određene na temelju veličine ikona i teksta. Manju ne.

Dodjeljivanje funkcionalnosti gumbima

Tužna vijest koju saznate neposredno prije nego što želite dodijeliti kakvu funkcionalnost gumbima u alatnoj traci jest da gumbi nemaju vlastite događaje, već se sve obavlja preko događaja vezanih uz samu alatnu traku. Preciznije rečeno, radi se o događaju `ButtonClick`, koji nastupi kada kliknemo na jedan gumb, bez obzira koji. Drugim riječima, zadatak nam je otkriti koji je gumb pritisnut i sukladno tome izvršiti adekvatnu funkcionalnost. Evo kako to izgleda u praksi:

```
private void toolBar1_ButtonClick(object sender,
    System.Windows.Forms.ToolBarButtonClickEventArgs e)
```

III. DIO: DIJELOVI .NET-A

```
{  
    if (e.Button == toolNovi)  
    {  
        menuNovi.PerformClick();  
    }  
    else if (e.Button == toolOtvori)  
    {  
        menuOtvori.PerformClick();  
    }  
    else if (e.Button == toolSnimi)  
    {  
        menuSnimi.PerformClick();  
    }  
}
```

Primijetite da se funkcija kreirana na temelju događaja ButtonClick malo razlikuje od većine ostalih funkcija vezanih uz neki događaj. Naime, drugi parametar nije tipa System.EventArgs, već ToolBarButtonClickEventArgs, što u praksi znači da osim standardnih svojstava sadrži i svojstvo Button, koje nam kaže koji je gumb kliknut.

Stoga u tijelu funkcije koristimo naredbu *if* iza koje uspoređujemo je li gumb sadržan u svojstvu Button jednak nekom od gumba na našoj alatnoj traci (izrazi toolNovi, toolOtvori i toolSnimi su imena gumba na traci). Ukoliko jest, znači da je kliknut upravo taj gumb pa u bloku koji slijedi možemo odrediti funkcionalnost tog gumba. Mi u našem primjeru koristimo funkcionalnosti definirane za stavke izbornika pa metodom PerformClick() simuliramo klik na određenu stavku.

Statusna traka

Dok je alatna traka uvijek smještena na vrhu forme, na njezinom dnu ćemo u većini programa naći statusnu traku. Ona je najčešće informativnog karaktera, a što će se na njoj naći ovisi o namjeni programa odnosno njegovu autoru. Mi smo u našem primjeru odlučili na ovo mjesto zapisivati ime datoteke koju uređujemo, broj znakova koje datoteka sadrži (što je ujedno i veličina datoteke u bajtovima) te obavijest o tome je li datoteka snimljena ili nije.

Statusnu traku možemo koristiti na dva načina. Prvi način je znatno jednostavniji – cijela će se traka tretirati kao jedno polje i u nju ćete moći umetnuti samo jedan podatak. U tom ćete slučaju podatak ubaciti tako da ga upišete u svojstvo Text i on će biti prikazan na traci.

Složena statusna traka je po mnogočemu slična traci s alatima. Ona je podijeljena na više dijelova: kao što alatna traka ima gumbе, tako ova ima panele u koje možemo smještati informacije koje želimo prikazati. Takvih panela u jednoj traci može biti više, onoliko koliko nam treba, no da bi bili vidljivi treba uključiti svojstvo ShowPanels.