

DODATAK **B**

Otklanjanje grešaka

U ovom poglavlju:

- Otklanjanje grešaka u programima uz pomoć Visual Studija .NET
- Korištenje prekidnih točki
- Praćenje poziva metoda u prozoru *Call Stack*
- Izvršavanje naredbi preko prozora *Command*
- Praćenje vrijednosti varijabli

Kao što ste mogli vidjeti u druženju s .NET-om kroz cijelu knjigu, greške su vrlo stvarne i ne događaju se nekom drugom. Na svu sreću, Visual Studio .NET nudi vam izvrsne mehanizme njihova pronalaženja, identificiranja i znatno olakšava njihovo otklanjanje.

U nastavku ovog poglavlja upoznat ćete osnovne načine *debugiranja* aplikacija odnosno upoznat ćete proces otklanjanja *bugova* i grešaka koji se obavlja detaljnim praćenjem rada aplikacije, promjene varijabli i slično. Sve što će biti prikazano u ovom poglavlju vezano je uz Visual Studio .NET, što znači da prikazane metode možete koristiti pri razvoju svih tipova aplikacija, od prozorskih preko web-aplikacija sve do web-servisa.

IV. DIO: DODACI

Praćenje rada aplikacije

Za upoznavanje s dijelovima sučelja koji pomažu pri pronalaženju i otklanjanju grešaka trebat će nam i program na kojem ćemo sve to testirati. Naravno, on će u sebi imati grešaka, no i dijelova kôda koji će poslužiti isključivo prikazivanju mogućnosti.

```
private void Form1_Load(object sender, System.EventArgs e)
{
    int a = 20;
    int b = 10;

    sqlDataAdapter1.Fill(dataSet11);

    if (b == 10)
    {
        b = b - 5;
    }

    double c = a / (b - 5);

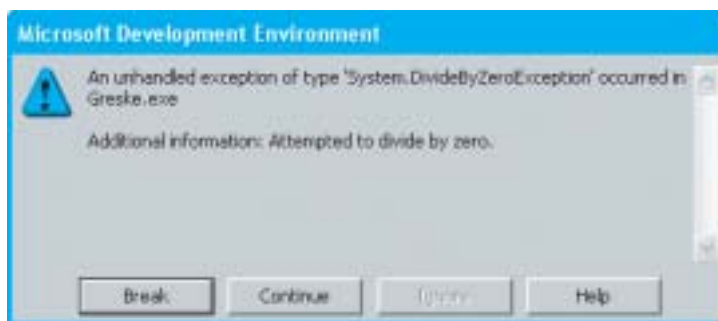
    textBox1.Text = c.ToString();
}
```

Nemojte da vas zbune objekti *DataAdapter* i *DataSet* – oni su prethodno definirani putem Visual Studio .NET sučelja i služit će da pokažemo neke mogućnosti sučelja. Isto tako, prethodna metoda je definirana u prozorskoj aplikaciji, no sve što će biti prikazano može biti primijenjeno i na druge tipove aplikacija.

U prethodnom kodu postoji očita greška – pri računanju varijable *c* pokušavamo dijeliti s nulom (što bi rezultiralo s beskonačnim brojem), što pri izvođenju aplikacije rezultira greškom na slici B-1.



Slika B-1:
Greška u aplikaciji kao
posljedica pokušaja dije-
ljenja s nulom



DODATAK B : OTKLANJANJE GREŠAKA

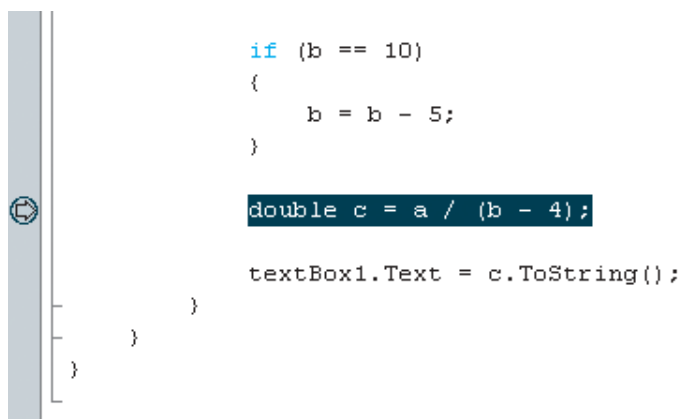
Da biste mogli *debugirati* aplikaciju, trebate je pokrenuti odabirom opcije *Debug – Start*. Ukoliko se aplikacija ispravno pokrene, možete se vratiti u Visual Studio – razvojna okolina će promijeniti svoje sučelje i pojaviti će se niz novih prozora, koje ćemo u nastavku objasniti.

Ukoliko ne želite svoje aplikacije pri pokretanju *debugirati*, mnogo je efikasnije pokrenuti opciju *Start Without Debugging* (CTRL+F5) jer će se tad aplikacija brže pokrenuti.



Prekidne točke

No za početak, potrebno je ukratko objasniti prekidne točke (engl. *breakpoint*) koje ćemo koristiti za kontrolu izvršavanja kôda. One omogućavaju zaustavljanje izvršavanja aplikacija kad se dođe do određene naredbe u kodu. Najjednostavniji način postavljanja prekidne točke je da kliknete na sivi rub uz liniju gdje je želite ubaciti. Na tom mjestu će se pojaviti crvena točka koja označava umetnutu prekidnu točku, a cijela naredba na kojoj se dešava prekid bit će obojana crvenom bojom.



Slika B-2:
Prekidna točka postavljena je na problematičnu naredbu.

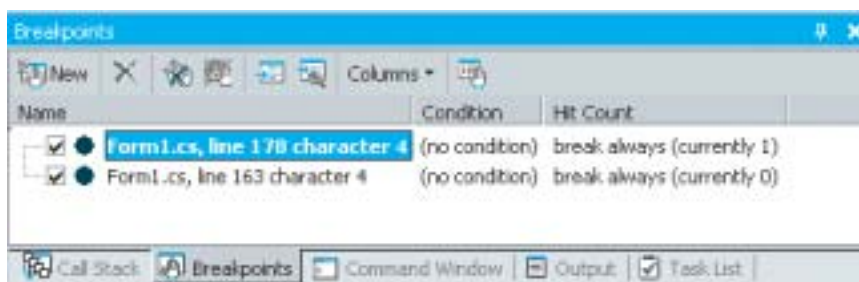
Prekidna točka zaustavlja izvršavanje programa netom prije naredbe u kojoj je ona postavljena.



IV. DIO: DODACI

Pokrenete li aplikaciju u načinu *Debug*, ona će se zaustaviti u zadanom trenutku, a fokus će se vratiti u Visual Studio .NET. Popis svih prekidnih točki možete vidjeti u prozoru *Breakpoints* prikazanom na slici B-3. Ukoliko ne vidite navedeni prozor, uključite ga opcijom *Debug – Windows – Breakpoints*.

Slika B-3:
Popis prekid-
nih točki u
programu
prikazan je u
prozoru
Breakpoints.



U programu možete imati proizvoljan broj prekidnih točki odnosno možete namjestiti da se izvršavanje aplikacije zaustavi na više različitih mjesta u programu.

Izvršavanje kôda naredbu po naredbu

Prekidne točke postavljat ćete u problematičnim dijelovima aplikacije, tj. u onima za koje sumnjate da rezultiraju greškom, no ne možete je otkriti. Nakon što zaustavite izvođenje aplikacije, iskoristit ćete različite dijelove sučelja koji će biti objašnjeni u nastavku poglavlja.

No prekidne točke mogu služiti i za kontrolu izvršavanja kôda. Pokrenemo li aplikaciju s prethodnim kôdom i postavljenom prekidnom točkom na *if*-naredbi, Visual Studio .NET će zaustaviti izvođenje aplikacije i prikazati žutu strelicu koja ukazuje na trenutno aktualnu naredbu (i sama naredba će biti u žutoj boji).

Slika B-4:
Izvršavanje
aplikacije zaustavl-
jeno je na prekidnoj
točki – žuta strelica
i boja označavaju
trenutno aktualnu
naredbu.



```
if (b == 10)
{
    b = b - 5;
}

double c = a / (b - 4);
```

DODATAK B : OTKLANJANJE GREŠAKA

U trenutku kad je zaustavljena aplikacija, na raspolaganju vam stoje tri opcije za kontrolu izvršavanja kôda – *Step Over*, *Step Out* i *Step Into*. Sve se nalaze u izborniku *Debug*.

Preporučljivo je naučiti tipkovničke kratice spomenutih opcija – *Step Over* možete izvršiti pritiskom na F10, *Step Into* pritiskom na F11, a *Step Out* pritiskom na SHIFT+F11. Korištenjem tih kratica mnogo ćete lakše kontrolirati izvršavanje kôda nego stalnim odlaskom u izbornik *Debug* ili klikanjem po plutajućem izborniku *Debug*.



Step Into vam služi za ulazak jednu razinu dublje u kôdu, ili barem prelazak na sljedeću naredbu. Tu ćete opciju koristiti za izvršavanje svake naredbe u kodu. Njenim odabirom izvršit će se trenutno aktualna naredba, a žuta oznaka će se postaviti na sljedeću naredbu. Ukoliko je sljedeći na redu za izvršavanje poziv neke metode, pokretanjem *Step Into* opcije ući ćete u tu metodu i izvršavati kompletan njen kôd naredbu po naredbu.

Suprotna od opcije *Step Into* je opcija *Step Over*. Ukoliko je sljedeći na redu poziv neke metode, pokretanjem ove opcije preskačete *debugiranje* te metode i svih njenih naredbi. No pripazite – to ne znači da se ta metoda neće izvršiti, već jednostavno nećete dobiti priliku za izvršavanje svake njene pojedine naredbe. Ta metoda će se odjedanput izvršiti, a za aktualnu naredbu bit će postavljena prva sljedeća naredba poslije poziva metoda. To je idealna opcija za slučaj kad ste sigurni da neka metoda ispravno radi i ne želite trošiti vrijeme na lupanje F10 za svaku njenu naredbu.

Step Out služi, kao što i ime kaže, za izlazak iz trenutne metode. No kao i kod opcije *Step Over*, to ne znači da se ostatak kôda metode koja se *debugira* neće izvršiti. Baš suprotno, on će se kompletno izvršiti, no vi nećete trebati ručno izvršavati svaku pojedinu naredbu. Dakle, ova je opcija idealna za situacije kad ste izvršili sve problematične naredbe u metodi i ostale su samo neke za koje ste sigurni da rade, a želite završiti s njenim *debugiranjem*. Odabirom ove opcije aktualna naredba za izvršavanje postaje sljedeća naredba nakon poziva metode koju ste *debugirali*.

Praćenje poziva metoda

Da biste se lakše snalazili s opcijom *Step Out*, na raspolaganju vam stoji prozor *Call Stack*. Ukoliko ga ne vidite pri *debugiranju* aplikacije, kliknite na *Debug – Windows – Call Stack*. Kao što mu i samo ime kaže, radi se o popisu poziva metoda predstavljenih u obliku stoga.

Stog je struktura organizirana po LIFO (*Last In, First Out*) principu, što znači da će element koji je zadnji dodan doći na početak strukture, ali će zato morati biti prvi maknut s nje. Element koji je prvi ušao u strukturu bit će na dnu i zato će zadnji moći biti s nje izbrisan.



IV. DIO: DODACI

Primijenite li navedenu definiciju stoga i njegove LIFO strukture bit će vam jasnije kako je organiziran prozor s popisom poziva metoda. Dakle, zamislite situaciju u kojoj imate metodu iz koje pozivate neku drugu metodu, a iz nje pak pozivate neku treću metodu, primjerice:

```
public void PrvaMetoda() {
    DrugaMetoda();
}

public void DrugaMetoda() {
    TrecaMetoda();
}

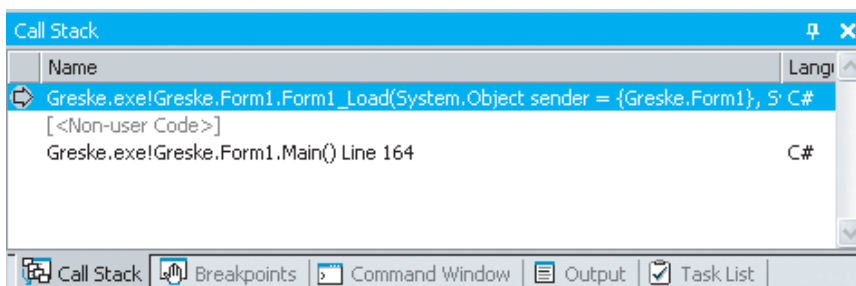
public void TrecaMetoda() {
    // neki kôd
}
```

Metoda koja je prva pozvana (recimo da je to *PrvaMetoda*) bit će na dnu liste. Iz nje je pak pozvana *DrugaMetoda* te je ona stavljena na mjesto na listi iznad nje. Iz nje je pak pozvana treća metoda, koja će tako biti stavljena na vrh liste, iznad druge dvije.

```
TrecaMetoda
DrugaMetoda
PrvaMetoda
```

Nakon što se završi izvršavanje treće metode, ona se miče s vrha liste i na njoj ostaju samo dvije metode, *PrvaMetoda* i *DrugaMetoda*. Nakon što se izvrši druga metoda, i ona se miče s liste i ostaje samo *PrvaMetoda*. Sad razumijete strukturu LIFO – *TrecaMetoda* je zadnja došla, no prva je otišla, jer je njeno izvršavanje prvo završilo.

Slika B-5:
Prikaz prozora
Call Stack –
metoda Main je
prva pozvana,
a zatim je poz-
vana metoda
Form_Load



Dakle, pri pozivanju opcije *StepOut* pogledom na prozor *Call Stack* uvijek možete znati u kojoj ćete metodi nastaviti s izvršavanjem svake pojedine naredbe – u onoj koja je navedena prva

ispod one u kojoj se trenutno nalazite. Kako će na vrhu prozora *Call Stack* biti navedena uvijek trenutna metoda, pozivanjem opcije *StepOut* nastavit ćete s izvršavanjem u drugoj metodi s liste, u naredbi koja dolazi odmah poslije poziva trenutne metode.

Prozor *Call Stack* može poslužiti i u složenim aplikacijama kako biste znali otkud je došao poziv trenutnoj metodi. Kliknete li na neku od metoda s popisa, u kodu će se pojaviti zelena oznaka koja će označavati trenutnu poziciju u toj metodi. Primjerice, ako istu metodu pozivate s više mjesta, klikom na metodu koja je ispod nje na listi u prozoru *Call Stack* možete doznati odakle je potekao njen poziv.

Izvršavanje naredbi za vrijeme izvršavanja aplikacije

Jedan od najkorisnijih prozora u procesu pronalaženja i otklanjanja grešaka je svakako prozor *Command*. Koliko ste puta u svom kodu htjeli izvršiti poziv neke metode samo da vidite da li ona vraća ispravnu vrijednost? Koliko ste puta u kodu htjeli promijeniti vrijednost neke varijable samo da vidite kako će dalje teći izvršavanje aplikacije? Ukoliko niste koristili prozor *Command*, bili ste osuđeni na mijenjanje kôda i dodavanje kôda za provjeru, kompajliranje cijele aplikacije i njeno ponovno pokretanje, što je bio ipak spor proces za jednostavna testiranja.

Korištenjem prozora *Command* možete izvršavati različite naredbe koje će direktno utjecati na izvršavanje kôda, odnosno rezultat će biti isti kao da su te naredbe izvedene u samom kodu aplikacije. Da biste prikazali prozor *Command*, ukoliko ga ne vidite kao jedan od ponuđenih prozora na dnu ekrana, odaberite njegovo prikazivanje iz izbornika *Debug – Windows*.

Da biste saznali vrijednost neke varijable u kodu, jednostavno upišite njeno ime u prozor *Command* i u sljedećem retku će se pojaviti njena vrijednost. Da biste promijenili njenu vrijednost, iskoristite znak jednakosti i upišite njenu novu vrijednost, koja čak može biti izračunata iz trenutnih vrijednosti drugih varijabli. U narednom prikazu kôda podebljano su prikazane naredbe koje su upisane u prozor *Command*, a normalnim stilom tekst koji je rezultat upisanih naredbi.

```
b
10
b=b+10
20
b=a+b
40
dataSet11.Employees.Rows.Count
9
```

Primijetite kako u prozoru *Command* možete ispisivati i propitkivati vrijednosti jednostavnih i složenih objekata, poput broja redaka u tablici *Employees* u objektu *dataSet11*. Imajte na umu da promjena vrijednosti svih varijabli direktno utječe na kôd – postavite li tako prekidnu točku oglednom

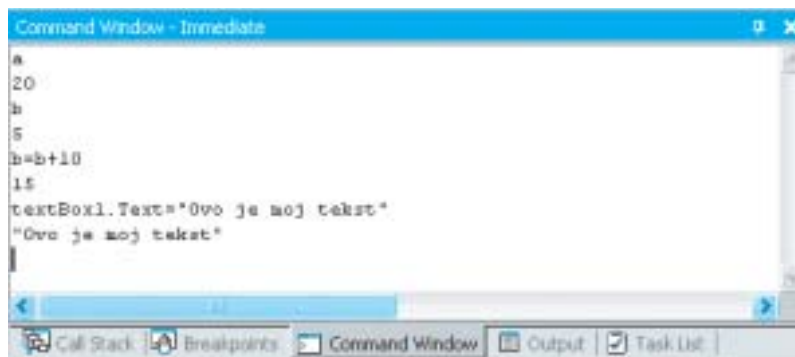
IV. DIO: DODACI

programu prije dijeljenja s nulom u varijabli *c* i u iskoristite prozor *Command* za mijenjanje vrijednosti varijable *b* (promijenite je da ne bude jednaka 5), to će direktno utjecati na njenu vrijednost u programu i dijeljenje s nulom se neće dogoditi.



Pri ispisivanju svojstava objekata u prozoru *Command*, kao i pri pisanju kôda, na raspolaganju vam stoji IntelliSense, koji će vam automatski ponuditi izbor svih dostupnih svojstava i metoda nekog objekta kad upišete točku iza njegova imena.

Slika B-6:
Korištenje prozora
Command može bitno
olakšati testiranje
rada aplikacije.



U prozoru *Command* mogli ste i pozivati različite metode i pratiti njihove rezultate. Pozivi metoda pišu se na isti način kao i u programu, a čak ćete uz pomoć tehnologije IntelliSense dobiti popis i opis parametara metode, što će vam dodatno olakšati testiranje.

Praćenje vrijednosti varijabli

Želite li na jednostavan način pratiti vrijednosti varijabli korištenih u programu, a da ne morate ručno svaki put upisivati njihovo ime u prozoru *Command*, na raspolaganju vam stoji prozor *Watch*. Dapače, zbog preglednosti i mogućnosti praćenja više različitih skupova varijabli, možete prikazati čak četiri prozora *Watch* – uključite ih preko opcija dostupnih u izborniku *Debug – Windows – Watch* ili pritiskom tipkovničke kratice CTRL+ALT+W u kombinaciji s jednim od brojeva između 1 i 4.

Da biste pratili vrijednost neke varijable, samo upišete njeno ime u stupcu *Name* u prozoru *Watch*. No ne samo što možete pratiti vrijednosti jednostavnih varijabli, već i objekata – upišete li ime nekog objekta, dobit ćete popis svih njegovih svojstava koje potom možete pratiti kroz izvršavanje aplikacije, a ona će zbog preglednosti biti uvučena i dostupna na klik na znak “+” kraj imena objekta.

DODATAK B : OTKLANJANJE GREŠAKA

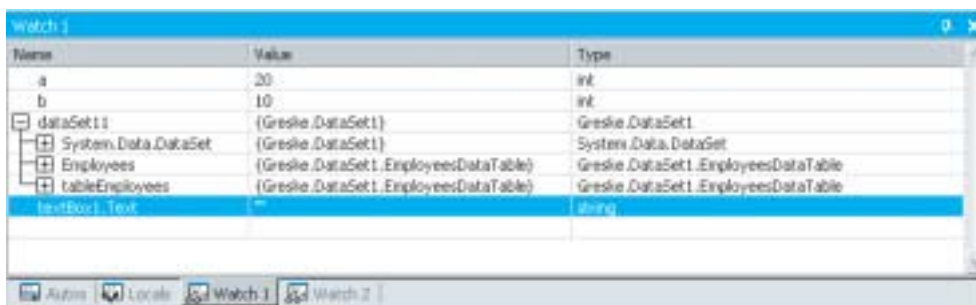
Preko prozora *Watch* također možete mijenjati vrijednosti svih prikazanih varijabli. Jednostavno kliknete u redak s varijablom u stupcu *Value* i upišite novu željenu vrijednost.



Korištenje prozora *Watch* idealno je za prvi korak u traženju greške – kad se uvjerite da su svi podaci ispravni i baš u obliku u kojem ih očekujete, svoju pažnju možete posvetiti programskoj logici.

Slika B-7:

Prozor *Watch* u akciji; uočite znak “+” pokraj pojedinih varijabli, što znači da one sadržavaju još čitav niz svojstava koja se mogu prikazati klikom na njega.



Prozor *Watch* će zapamtiti koje ste varijable upisali u njega i pri svakom sljedećem pokretanju aplikacije u *Debug* načinu rada one će biti prikazane u njemu, što znatno olakšava praćenje rada aplikacije. Stanja varijabli u prozoru *Watch* najbolje je pratiti korištenjem opisanih metoda za izvršavanje kôda naredbu po naredbu, jer tako imate potpunu kontrolu i nikakva vam promjena ne može umaći. Dapače, promjena vrijednosti varijabli koja se obavila u prethodnoj naredbi je u prozoru *Watch* označena crvenom bojom.

Uz prozor *Watch* u istoj skupini je i prozor *Autos* koji također služi za prikaz stanja varijabli, no u njemu se automatski prikazuju samo varijable koje su povezane s trenutno aktualnom linijom. Ukoliko niste prezahtjevni, pogledom na ovaj prozor uvijek možete vidjeti stanja najvažnijih varijabli koje se koriste pri izvođenju neke naredbe, što bi vam u većini slučajeva moglo biti dosta za pregled rada aplikacije.



IV. DIO: DODACI

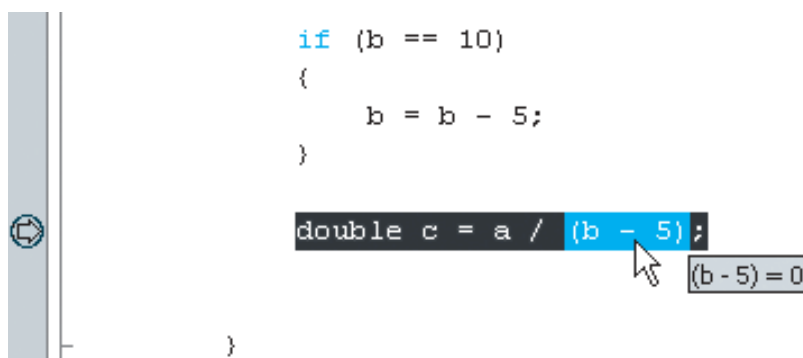
Quick Watch

Osim praćenja stanja varijabli preko posebnog prozora *Watch*, na raspolaganju vam stoji i još jedna vrlo korisna funkcionalnost Visual Studija .NET nazvana *Quick Watch*. Naime, u procesu *debugiranja* možete na vrlo jednostavan način saznati vrijednost varijabli ili nekog izraza.

Trebate samo privući miša varijabli u kodu i iznad nje će se pojaviti njena vrijednost. Tako se možete mišem kretati po cijelom kodu aplikacije i propitkivati varijable o njihovoj vrijednosti.

No to nije sve – kao što smo rekli, možete pratiti i vrijednost nekog izraza, kao što je prikazano na slici B-8. Trebate samo označiti željeni izraz, a njegova vrijednost će vam se također pojaviti u oblačiću.

Slika B-8:
Označavanjem
nekog izraza
Visual Studio će
prikazati njegovu
vrijednost.

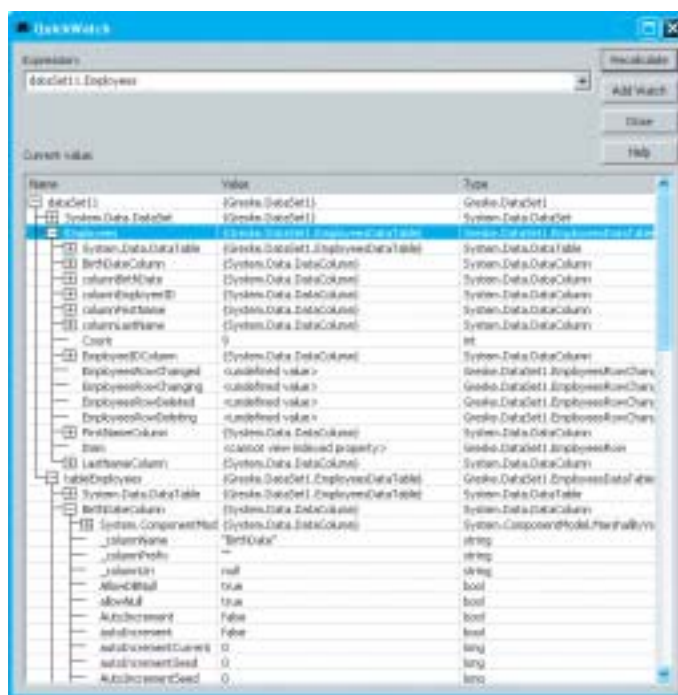


Kako se u oblačiću mogu prikazati samo jednostavne vrijednosti varijabli, što je prilično ograničenje ukoliko želite ispitivati svojstva objekata i svih njihovih vrijednosti, možete iskoristiti opciju *Quick Watch*. Kliknete li tako na neki objekt čiju vrijednost ne možete prikazati u jednostavnom oblačiću (primjerice, objekt *DataSet*) desnim gumbom miša, u izborniku možete odabrati opciju *Quick Watch*.

Praćenje rada aplikacije ispisivanjem poruka

Najjednostavniji je način praćenja rada aplikacije i – jer u nekom od svojih oblika postoji od prvih metoda programiranja – najpopularniji, ispisivanje različitih poruka u pojedinim dijelovima aplikacije. Najčešće su to jednostavne poruke koje trebaju svojim pojavljivanjem samo dati do znanja da se određeni dio kôda izvršio pa se ispisuju poruke poput “tu sam”, “radi” i slično. Naravno, često se i u takvom ispisivanju poruka ispisuju i vrijednosti nekih varijabli, no to ipak više ne morate raditi jer vam na raspolaganju stoje prozor *Watch* i opcija *Quick Watch*.

DODATAK B : OTKLANJANJE GREŠAKA



Slika B-9:
Detaljan prikaz nekog objekta tipa *DataSet* u prozoru *Quick Watch*

No zahvaljujući Visual Studiju više niste osuđeni na ispisivanje poruka koje se pojavljuju u samoj aplikaciji (bilo kao neki tekst na web-stranici u web-aplikacijama ili kao poruka prikazana metodom *MessageBox.Show* u prozorskim aplikacijama). U Visual Studiju možete različite poruke ispisivati u prozoru *Output*, i to korištenjem *Debug.WriteLine* naredbe koja radi u svim tipovima aplikacija, jer nije vezana uz samo sučelje aplikacije, već uz razvojnu okolinu.

Da biste mogli koristiti klasu *Debug* i njenu metodu *WriteLine* (kao i druge metode iz klase), u svoju aplikaciju trebate uključiti *namespace System.Diagnostics*.



Evo kako možete koristiti klasu *Debug* za ispisivanje poruka:

```
Debug.WriteLine("Korak 2 izvršen!");
```

Često se ispisivanje tih poruka koristi i u *if*-naredbama jer se želi ispisati posebna poruka ako se ušlo u *if*-blok, odnosno ako se ušlo u *else*-blok (tj. želi se saznati da li je uvjet u *if*-naredbi ispunjen).

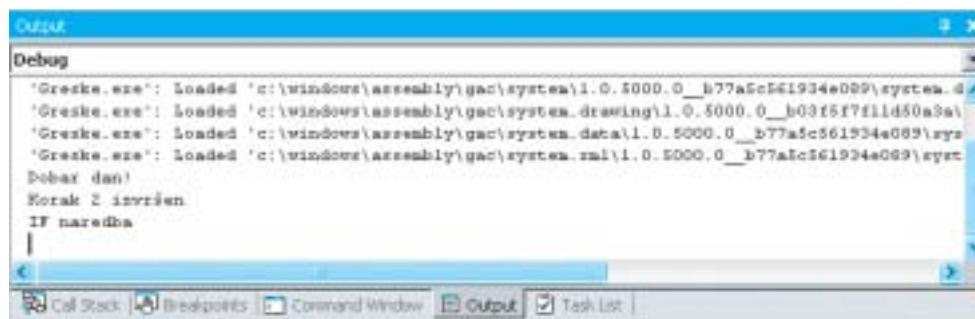
IV. DIO: DODACI

Klasa *Debug* nudi vam jednostavniji način – možete iskoristiti metodu *WriteLineIf* u kojoj kao prvi parametar navodite uvjet koji treba provjeriti. Ukoliko je taj uvjet istinit, poruka će se ispisati, a u protivnom, naravno, neće.

```
Debug.WriteLineIf(a > b, "A je veći od B!");
Debug.WriteLineIf(textBox1.Text != "", "Nešto piše u TextBoxu!");
```

Slika B-10:

Poruke ispisane metodama klase *Debug* pojavit će se u prozoru *Output*.



U prozoru *Output* ispisuju se i poruke vezane uz izvršavanje programa (primjerice, o učitanim *assemblyjima*).

Korištenjem ovdje opisanih opcija možete značajno olakšati razvoj aplikacija. Greške su nešto normalno i očekivane su pri razvoju i najjednostavnijih aplikacija. Stoga ne treba stavljati naglasak na izradu savršenog kôda bez grešaka (zar to uopće postoji?), već na pronalaženje problematičnih dijelova kôda i traženju grešaka. Zahvaljujući ovdje opisanim metodama to više nije bauk i predstavlja izrazito jednostavan i moćan proces koji možete koristiti u svakoj fazi izrade aplikacija. Što se bolje upoznate s mogućnostima razvojne okoline Visual Studija, izrada aplikacija bit će vam lakša i ugodnija.