C#

Izvornik

Stanley B. Lippman

Prevod
Jan Sokol

CET Computer Equipment and Trade
Beograd

Addison-Wesley

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

C# Izvornik

ISBN 86-7991-201-8

Autorizovan prevod sa engleskog jezika knjige **C# Primer : A Practical Approach**

PUBLISHED BY Addison-Wesley, USA.

Prevod	Jan Sokol
Recenzent	Nikola Milanović
Lektor	Mirjana Terzić
Urednik	Radmila Ivanov
Tehnički urednik	Dušan Čašić
Prelom	Zoran Stevanović
Izdavač	CET Computer Equipment and Trade Beograd, Skadarska 45 tel/fax: 011 3243-043, 3235-139, 3237-246 http://www.cet.co.yu e-mail: redakcija@cet.co.yu
Za izdavača	Dragan Stojanović, direktor
Obrada korica	Bit Inženjering, Beograd
Tiraž	1000
Štampa	„Svetlost", Čačak

Sadržaj

Predgovor

C# je novi jezik napravljen u Microsoftu i predstavljen sa Visual Studio .NET-om. Više od milion redova C# koda je već upotrebljeno za implementaciju klasa .NET okruženja. Ova knjiga predstavlja jezik C# i njegovu upotrebu u programiranju klasa .NET okruženja ilustrujući aplikacione domene kao što su ASP.NET i XML.

Najpre ću predstaviti programski zadatak, a onda prikazati jednu ili dve implementacije opisujući jezička svojstva ili aspekte klase okruženja. Cilj je da pokažem kako koristiti jezik i klase okruženja za rešavanje problema, a ne da jednostavno nabrojim osobine jezika i API klasa okruženja.

Učenje jezika C# je dvostepeni proces: učenje jezika C# i upoznavanje klasa .NET okruženja. Ovaj dvostepeni proces se ogleda u organizaciji ovog teksta.

U prvom koraku prikazujemo jezik – njegove mehanizme kao što su klase i nasleđivanje interfejsa i delegati, i odgovarajuće koncepte kao što je unificirani sistem tipova, referentni naspram vrednosnih tipova, *pakovanje* i tako dalje. Ovaj korak je objašnjen u prva četiri poglavlja.

Drugi korak je upoznavanje sa klasama .NET okruženja, posebno sa Windows i Web programiranjem, kao i podrškom za XML. Ova tema je obrađena u drugoj polovini knjige.

Ako savladate sve lekcije iz ove knjige, vaše veštine C# programiranja bi trebalo da postanu mnogo bolje.. Pored toga, upoznaćete se sa dobrim delom klasa .NET okruženja. Sav kod je dostupan za preuzimanje na Web lokaciji moje kompanije www.objectwrite.com.

Poštu mi možete poslati direktno na slippman@objectwriter.com.

Organizacija knjige

Knjiga sadrži osam relativno dugih poglavlja. Prva četiri poglavlja su posvećena jeziku C# i u njima se opisuju ugrađene jezičke osobine, mehanizam klasa, nasleđivanje klasa i nasleđivanje interfejsa. Sledeća četiri poglavlja istražuju različite domene biblioteka podržanih u klasama .NET okruženja.

U poglavlju 1 opisuju se osnove jezika, kao i neke osnovne klase obezbeđene u klasama okruženja. Ovo poglavlje vođeno je dizajniranjem male aplikacije. Uvedeni su koncepti kao što su prostori imena, rukovanje izuzecima i unificirani sistem tipova.

U poglavlju 2 opisuju se osnove pravljenja klasa. Naučićemo šta je nivo pristupa, koje su razlike između članova `const` i `readonly` i objasniti specijalizovane metode kao što su indekseri i svojstva. Opisaćemo različite strategije inicijalizacije članova, kao i pravila za preopterećivanje operatora i operatore konverzije. Naučićemo šta je to tip `delegate` koji služi kao vrsta univerzalnog pokazivača na funkciju.

U poglavljima 3 i 4 prikazuju se nasleđivanje klase i interfejsa. Nasleđivanje klase nam omogućava da definišemo familiju specijalizovanih tipova koji nadjačavaju generički interfejs, kao što je apstraktna bazna klasa `WebRequest` i protokolno specifični podtip `HttpWebRequest`. Sa druge strane, nasleđivanje interfejsa nam omogućava da obezbedimo zajednički servis ili deljene atribute za inače nepovezane tipove. Na primer, interfejs `IDisposable` oslobađa resurse. Klase koje drže veze sa bazom podataka ili rukovaoce prozorom će najverovatnije implementirati `IDisposable` iako su na drugi način nepovezane.

Poglavlje 5 obezbeđuje širok obim biblioteka klasa u .NET-u. Naučićemo šta je to ulaz i izlaz, uključujući rukovanje datotekom i direktorijumima, regularne izraze, sokete i programiranje u nitima, hijerarhije klase `WebRequest` i `WebResponse`, kratki uvod u ADO.NET i ostvarivanje veze sa bazom podataka i upotrebu XML-a.

Poglavlja 6 i 7 objašnjavaju "povuci i pusti" razvoj Windows obrazaca i Web obrazaca. Poglavlje 7 se usredsređuje na ASP.NET i životni ciklus Web stranice. Oba poglavlja obezbeđuju mnogo primera upotrebe postojećih kontrola i dodavanje rukovaoca događajima za korisničku interakciju.

Poslednje poglavlje obezbeđuje programerski uvod u .NET zajednički izvršni jezik (Common Language Runtime). Ono daje fokus na sklopove, refleksiju tipova, atribute i završava se kratkim pregledom odgovarajućeg međujezika koji je cilj kompilacije svih .NET jezika.

Napisano za programere

U ovoj knjizi se ne očekuje da poznajete C++, Visual Basic ili Javu, ali se pretpostavlja da ste programirali u nekom jeziku. Ne očekujem, na primer, da znate tačnu sintaksu naredbe petlje `foreach` jezika C#, ali pretpostavljam da znate šta je to petlja. Iako ću ilustrovati kako da pozovete funkciju u jeziku C#, pretpostavljam da znate šta mislim kada kažem `"pozvati funkciju"`. Ovaj tekst ne zahteva prethodno znanje objektno orijentisanog programiranja ili ranije verzije ASP-a i ADO-a.

Neki ljudi – neki vrlo pametni ljudi – smatraju da je pod .NET-om programski jezik na drugom mestu u odnosu na odgovarajući zajednički izvršni jezik (Common Language Runtime – CLR) oko kojeg jezici plutaju kao kontinenti na tektonskim pločama. Ja se ne slažem. Jezik je način kako se izražavamo, i izbor jednog jezika utiče na dizajn naših programa. U ovoj knjizi se pretpostavlja da je jezik C# omiljeni jezik za .NET programiranje.

Knjiga je organizovana u osam relativno dugih poglavlja. Prva četiri poglavlja se usredsređuju na jezik C# pregledajući ugrađene jezičke osobine, mehanizam klasa, nasleđivanje klasa i nasleđivanje interfejsa. Sledeća četiri poglavlja istražuju različite domene biblioteka podržanih u klasama .NET okruženja, kao što su regularni izrazi, niti, soketi, Windows obrasci, ASP.NET i zajednički izvršni jezik.

Leksičke konvencije

Imena tipova, objekti i ključne reči su u fontu Courier kao što je `int`, unapred definisani tip u jeziku; `Console`, klase definisane u okruženju; `maxCount`, objekat definisan ili kao podatak član ili kao lokalni objekat u funkciji; i `foreach`, jedna od unapred definisanih naredbi petlji. Iza imena funkcija sledi par praznih zagrada kao u `WriteLine()`. Prvo uvođenje koncepta, kao što je *sakupljanje otpadaka* ili *enkapsuliranje podataka,* predstavljen je kurzivnim fontom. Namera ovih konvencija je da tekst učine čitljivijim.

Priznanja

Ova knjiga je rezultat rada mnogih koji su pomagali njenom autoru da ostane na kursu. Najsrdačnije zahvaljujem svojoj ženi Bet i svojoj deci, Danijelu i Ani. Propustio sam mnogo porodičnih izlazaka da bi ova knjiga mogla biti urađena. Hvala svima što ste bili (uglavnom) strpljivi i puni razumevanja i niste prečesto pitali da li sam već završio.

Treba da zahvalim Caro Segal i Shimon Cohenu na you-niversity.com, koji su mi posvetili mnogo vremena i pružili ohrabrenje. Takođe dugujem zahvalnost Ericu Gunnersonu, Peteru Draytonu i Donu Boxu, koji su u jednom ili drugom trenutku ispunjavali ulogu viteza na konju.

Želeo bih da se zahvalim i Eleni Driskill. Dva puta. Prvo, za one divne crteže u poglavlju 6. Drugo, za njenu dozvolu da ih reprodukujem.

Deborah Lafferty je bila moj urednik od prvog izdanja *C++ Primer* davne 1986. Ona je uvek imala razumevanja za mene i ja veoma cenim njena ohrabrivanja (i podsticanja) u realizaciji ovog projekta.

Nekoliko specijalnih produkcijskih zahvalnica upućujem Stephanie Hiebert i Steve Hallu. Stephanie je glavni urednik izdanja mog skoro dvadesetogodišnjeg izdavanja. Ona je ovu knjigu učinila boljom. Steve me je vratio u moje sedlo za kucanje pošto sam bio uzdrman široko opasnim Framemaker problemima. Neka deo mog virtuelnog šešira dobijete oboje.

Sledeći saradnici su ponudili mnoge pametne komentare i sugestije pri pregledanju različitih delova ovog rukopisa: Indira Dhingra (posebno hvala za obezbeđivanje poslednjeg smisaonog pregleda ovog rukopisa), Cay Horstmann, Eugene Kain, Jeff Kwak, Michael Lierheimer, Drew Nathanson, Clovi Tondo i Damien Watkins.

Delovi ovog rukopisa su bili isprobani na kursevima i razgovorima širom naše planete: Sydney, Amsterdam, Munich, Tel Aviv, Orlando, San Francisco i San Jose. Hvala svima koji su obezbedili povratni odgovor.

Resursi

Najbogatija dokumentacija kojoj ćete se vremenom vraćati opet je Visual Studio .NET dokumentacija. Reference .NET okruženja su osnova za pravljenje bilo kog C#/.NET programiranja.

Drugi bogati izvor o .NET-u sastoji se od artikala i članaka u *MSDN Magazinu*. Uvek sam impresioniram onim što nađem u svakom od njih. Možete ga naći na adresi http://msdn.microsoft.com/msdnmag.

Diskusiona lista DOTNET, koju sponzoriše DevelopMentor, bogat je izvor informacija. Na nju se možete prijaviti na adresi http://discuss.develop.com.

Sve što su Jeffrey Richter, Don Box, Aaron Skonnard ili Jeff Prosise napisali o .NET-u (ili XML-u u slučaju Aarona) treba obavezno pročitati. Trenutno, najveći deo onog što pišu pojavljuje se u *MSDN Magazinu*.

Navodim spisak knjiga za koje mislim da su veoma korisne:

- *Active Server Pages+*, Richard Anderson, Alex Homer, Rob Howard i Dave Sussman, Wrox Press, Birmingham, England, 2000.
- *C# Essentials*, Ben Albahari, Peter Drayton i Brad Merrill, O'Reilly, Cambridge, MA, 2001.
- *C# programming,* Burton Harvey, Simon Robinson, Julian Templeman, i Karli Watson, Wrox Press, Birmingham, England, 2000.
- *Essental XML: Beyond Markup*, Don Box, Aaron Skonnard i John Lam, Addison-Wesley, Boston, 2000.
- *Microsoft C# Language Specifications*, Microsoft Press, Redmond, WA, 2001.
- *A Programmer's Introduction to C#, 2nd Edition*, Eric Gunnerson, Apress, Berkeley, CA, 2001.

Stanley Lippman
Los Angeles
November 18, 2001.
www.objectwrite.com

Poglavlje 1

Zdravo C#

Moja ćerka je učila da svira brojne muzičke instrumente. Sa svakim od njih je imala strah da počne svirati klasiku – ne, ne Schubert ili Schoenberg, nego Backstreet Boys i Britney Spears. Trudeći se da zadrže njeno interesovanje dok su je upućivali u osnove sviranja, njeni učitelji muzike su joj uglavnom povlađivali. U nekom smislu ovo poglavlje pokušava da održi isti nesigurni balans u predstavljanju jezika C#. U našem slučaju klasika je predstavljena Web obrascima i nasleđivanjem tipova. Osnove su naoko prizemni unapred definisani jezički elementi i mehanizmi, kao što su pravila opsega, aritmetički tipovi i prostori imena. Moj metod podrazumeva da uvodim jezičke elemente kako postaju neophodni za implementaciju prvog malog programa. Za one koji razmišljaju tradicionalnije, poglavlje se završava zaključnom listom unapred definisanih jezičkih elemenata.

C# podržava oba numerička tipa, celobrojni i sa pokretnim zarezom, tip Boolean, tip Unicode znak i visoko precizne decimalne tipove. Ovi tipovi se posmatraju kao *jednostavni tipovi*. Sa ovim tipovima je povezan skup operatora, uključujući sabiranje (+), oduzimanje (–), jednakost (==) i nejednakost (!=). C# takođe obezbeđuje unapred definisani skup naredbi kao što su uslovne `if` i `switch` naredbe, kao i naredbe za izvršavanje petlji `for`, `while` i `foreach`. Sve one, kao i mehanizmi prostora imena i rukovanja izuzecima, opisani su u ovom poglavlju.

1.1 Prvi C# program

Tradicionalni prvi program u novom jeziku je onaj koji na korisničkoj konzoli štampa Hello, World! U C#-u ovaj program se implementira ovako:

```
// Naš prvi C# program
using System;
class Hello
{
      public static void Main()
      {
            Console.WriteLine( "Hello, World!" );
      }
}
```

Kada se kompajlira i izvrši, naš program generiše poznato

```
Hello, World!
```

Naš program ima četiri elementa: (1) komentar predstavljen dvostrukom kosom crtom (//), (2) direktivu `using`, (3) definiciju klase i (4) funkciju članica klase (alternativno nazvan metod klase) koja je nazvana `Main()`.

C# program počinje izvršavanje u funkciji `Main()`, članici klase. Ovo se naziva ulazna tačka programa. `Main()` mora biti definisana kao `static`. U našem primeru deklarišemo je i kao `public` i kao `static`.

`public` identifikuje dozvoljeni nivo pristupa funkciji `Main()`. Članu klase deklarisanom kao `public` može se pristupiti iz bilo kog dela programa. Član klase generalno je ili funkcija članica koja izvršava određenu operaciju povezanu sa ponašanjem klase, ili podatak član koji sadrži vrednost povezanu sa stanjem klase. Tipično, funkcije članice klase se deklarišu kao `public`, a članovi podaci se deklarišu kao `private`. (Nivoe pristupa članu opet ćemo razmotriti kada budemo dizajnirali klase.)

Generalno, funkcije članice klase podržavaju ponašanje povezano sa klasom. Na primer, `WriteLine()` je javna funkcija članica klase `Console`. `WriteLine()` štampa rezultat na korisničku konzolu, posle kojeg sledi znak za novi red. Takođe, klasa `Console` obezbeđuje funkciju `Write()`. `Write()` štampa izlaz na terminal, ali bez umetanja znaka za novi red. Tipično, `Write()` koristimo kada želimo da korisnik odgovori na upit prosleđen konzoli, a `WriteLine()` kada jednostavno prikazujemo informaciju. Ubrzo ćemo videti relevantan primer.

Primarna aktivnost C# programera je da dizajniraju i implementiraju klase. Šta su klase? Uglavnom predstavljaju entitete u našem aplikacionom domenu. Na

primer, ako razvijamo bibliotečki sistem za izdavanje knjiga, verovatno će nam biti potrebne klase kao što su `Book`, `Borrower` i `DueDate` (sa aspekta vremena).

Odakle dolaze klase? Naravno, najčešće od programera kao što smo mi. Ponekad je naš posao da ih implementiramo. Ova knjiga je osmišljena i napravljena tako da od vas napravi eksperta u toj oblasti. Ponekad su klase već dostupne. Na primer, .NET System okruženje obezbeđuje klasu `DateTime` koja je podesna za korišćenje u našem predstavljanju apstrakcije `DueDate`. Jedan od izazova da se postane ekspert C# programer – i to ne trivijalan – je mogućnost bliskosti sa više od 1000 klasa definisanih unutar .NET okruženja. Ne mogu sve da ih obuhvatim u ovom tekstu, ali ćemo razmotriti dobar broj klasa uključujući podršku za regularne izraze, niti, sokete, XML i Web programiranje, podršku za baze podataka i novi način građenja Windows aplikacija.

Izazovan problem je kako logično organizovati hiljadu ili više klasa tako da korisnici (to smo mi) mogu locirati i od njih napraviti neki smisao (i sačuvati imena od kolizije sa drugim imenima). Fizički ih možemo organizovati unutar direktorijuma. Na primer, sve klase koje podržavaju `Active Server Pages` (ASP) mogu se staviti u `ASP.NET` direktorijum pod korenskim direktorijumom `System.NET`. Ovo organizaciju čini sasvim jasnom nekome ko pretura po strukturi direktorijuma.

Šta sa unutrašnjostima programa? Kako se ispostavilo, postoji analogno organizovan mehanizam unutar samog C#-a. Radije nego da definišemo fizički direktorijum, identifikujemo *prostor imena* (*namespace*). Za .NET okruženje, prostor imena koji se nalazi na vrhu nazvan je `System`. Na primer, klasa `Console` je definisana unutar prostora imena `System`.

Grupi klasa koje podržavaju zajedničku apstrakciju je dat njihov vlastiti prostor imena definisan unutar prostora imena `System`. Na primer, prostor imena `Xml` je definisan unutar prostora imena `System`. (Kažemo da je prostor imena `Xml` ugnežđen unutar prostora imena `System`.) Zauzvrat, nekoliko prostora imena je ugnežđeno unutar prostora imena `Xml`. Na primer, postoji prostor imena `Serialization`, kao i prostori imena `XPath`, `Xsl` i `Schema`. Ovi razdvojeni prostori imena unutar prostora imena `Xml` su razdvojeni da bi enkapsulirali i lokalizovali deljenu funkcionalnost unutar opšteg opsega XML-a. Na primer,

ovo uređenje olakšava identifikaciju podrške koju .NET obezbeđuje za World Wide Web Consortium (W3C) `XPath` preporuku. Drugi prostori imena, ugnežđeni unutar prostora imena `System`, uključuju `IO` koji sadrži klase za datoteke i direktorijume, `Collections`, `Threading`, `Web` i tako dalje.

U strukturi direktorijuma pokazujemo vezu sadržanih direktorijuma i onog koji ih sadrži sa obrnutom kosom crtom (\), bar pod Windowsom – na primer,

```
System\Xml\XPath
```

Sa prostorima imena, slično, vezu između sadržanih prostora i onog koji ih i sadrži prikazujemo pomoću operatora scope (.) na mestu obrnute kose crte – na primer:

```
System.Xml.XPath
```

U oba slučaja znamo da je `XPath` sadržan unutar `Xml`, koji je sadržan unutar `System`.

Kad god se pozovemo na ime u C# programu, kompajler mora povezati to ime sa stvarnom deklaracijom nečega negde u našem programu. Na primer, kada napišemo:

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

kompajler mora nekako otkriti da je `Console` ime klase i da je `WriteLine()` funkcija članica unutar klase `Console` – što znači da je unutar opsega definicije klase `Console`. Zato što smo u našoj datoteci definisali samo klasu `Hello`, bez naše pomoći kompajler ne može razrešiti na šta upućuje ime `Console`. Kad god kompajler ne može da razreši na šta upućuje ime, on generiše grešku za vreme kompajliranja, što zaustavlja građenje našeg programa:

```
C:\C#Programs\hello\hello.cs(7):

    The type or namespace name 'Console' does
    not exists in the class or namespace

    (Tip ili prostor imena 'Console' ne postoji
    u klasi ili prostoru imena)
```

Direktiva using u našem programu,

```
using System;
```

upućuje kompajler da pogleda u prostor imena `System` za bilo koje ime koje ne može odmah razrešiti unutar datoteke koja se obrađuje – u ovom slučaju, datoteku koja sadrži definiciju naše klase `Hello()` i njenu funkciju članicu `Main()`.

Alternativno, možemo kompajleru eksplicitno reći gde da traži:

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

Neki ljudi – u stvari, neki pametni i u svakom slučaju vrlo pristojni ljudi – veruju da je eksplicitno izlistavanje *potpuno kvalifikovanih imena* – što znači, ono koje identifikuje ceo skup prostora imena u kojima je klasa sadržana – uvek bolje od direktive `using`. Oni naglašavaju da potpuno kvalifikovano ime jasno identifikuje gde se klasa nalazi i veruju da je ovo korisna informacija (čak i ako je ponovljena 14 puta unutar 20 susednih redova). Ne delim njihovo mišljenje (i stvarno ne volim toliko kucanje). U mom tekstu – i ovo je jedan od razloga zašto mi, autori, pišemo knjige – potpuno kvalifikovano ime klase se nikad ne koristi[1], osim da se razreši dvosmislena upotreba imena tipa (pročitajte odeljak 1.2 za ilustraciju situacija u kojima je ovo neophodno).

Ranije sam napisao da klase najčešće dolaze od drugih programera ili iz biblioteka koje obezbeđuje razvojni sistem. Odakle još dolaze? Iz samog C# jezika. C# unapred definiše nekoliko najčešće korišćenih tipova podataka, kao što su celi brojevi, jednostruko i dvostruko precizni tipovi sa pokretnim zarezom, i stringovi. Svaki tip ima dodeljeni specifikator tipa koji identifikuje tip u C#-u: `int` predstavlja primitivni tip celog broja, `float`, primitivni tip sa jednostrukom preciznošću; `double` tip sa dvostrukom preciznošću; i `string`, tip niza znakova. (Obratite pažnju na tabele 1.2 i 1.3 u odeljku 1.18.2 za listu unapred definisanih numeričkih tipova.)

Na primer, alternativna implementacija našeg jednostavnog programa definiše objekat `string` i inicijalizuje ga string literalom `"Hello, World!"`.

```
public static void Main() {
    string greeting = "Hello, World!";
    Console.WriteLine( greeting );
}
```

1. Čarobnjaci Visual Studia, kao što su Windows Forms i Web Forms, generišu potpuno kvalifikovana imena. Međutim, zato što su imena mašinski generisana, ne kvalifikuju se kao primeri za brojanje.

`string` je ključna reč jezika C# – a to je reč rezervisana za jezik C# i uvedena sa specijalnim značenjem. `public`, `static` i `void` su takođe ključne reči jezika. `greeting` se posmatra kao *identifikator*. On obezbeđuje ime za objekat tipa `string`. Identifikatori u jeziku C# moraju počinjati ili znakom podvlačenja (_) ili abecednim karakterom. Imena su osetljiva na razliku u velikim i malim slovima, tako da `greeting`, `Greeting` i `Greeting1` predstavljaju posebne identifikatore.

Zajednička tačka rasprave među programerskim centrima je da li složena imena treba da budu razdvojena znakom podvlačenja, kao na primer `xml_text_reader`, ili kapitalizacijom prvog slova svake unutrašnje reči, kao na primer `xmlTextReader`. Po konvenciji, identifikatori koji predstavljaju imena klasa uglavnom počinju velikim slovom, kao `XmlTextReader`.

Unutar jedinice programske vidljivosti, koja se još naziva *prostor deklaracija* ili *opseg*, identifikatori moraju biti jedinstveni. U *lokalnom opsegu* – a to je, u telu funkcije, kao u našoj definiciji funkcije `Main()` – ovo nije problem zato što generalno kontrolišemo celu definiciju bilo kog objekta u funkciji. Kako se područje opsega širi – odnosno, kako se povećava broj uključenih programera i organizacija – problem jedinstvenih identifikatora postaje sve teži. Ovo je mesto gde prostori imena stupaju na scenu.

1.2 Prostori imena

Prostori imena su mehanizam za kontrolisanje vidljivosti imena unutar programa. Oni su namenjeni da olakšaju kombinovanje programskih komponenti iz različitih izvora tako što minimizuju konflikte između imena identifikatora. Pre nego što saznamo više o mehanizmu prostora imena, uverimo se da razumemo problem radi čijeg rešenja su uvedeni prostori imena.

Imena koja nisu smeštena u prostor imena se automatski stavljaju u jedan neimenovani globalni prostor deklaracija. Ova imena su vidljiva u programu bez obzira da li se javljaju u istoj ili odvojenoj programskoj datoteci. Da bi se program izgradio, svako ime u globalnom prostoru deklaracija mora biti jedinstveno. Globalna imena otežavaju uključivanje nezavisnih komponenti u programe.

Na primer, zamislite da razvijate dvodimenziionalnu (2D) grafičku komponentu i da nazovete jednu od globalnih klasa imenom `Point`. Koristite komponen-

tu i sve radi odlično. O tome kažete nekom od prijatelja i oni naravno žele takođe da je koriste.

U međuvremenu, ja sam razvio trodimenzionalnu grafičku komponentu i jednu od globalnih klasa nazvao imenom `Point`. Koristim komponentu i takođe sve radi odlično. Pokažem je nekolicini mojih prijatelja. Komponenta im se sviđa i oni žele takođe da je koriste. Za sada su svi srećni bar oko projekata koje kodiramo.

Sada zamislite da imamo zajedničkog prijatelja. Ona implementira 2D/3D igru i želi da upotrebi naše dve komponente, obe visoko cenjene. Nažalost, kada uključi obe komponente u aplikaciju, dve nezavisne upotrebe identifikatora `Point` rezultiraju greškom za vreme kompajliranja. Njena igra ne uspeva da se izgradi. Pošto ona nema nijednu komponentu, ne postoji jednostavna ispravka da bi dve komponente radile zajedno.

Prostori imena obezbeđuju opšte rešenje za problem sukoba globalnih imena. Prostor imena daje ime unutar kojeg su klase i drugi tipovi enkapsulirani[2]. To znači da imena koja se nalaze u prostoru imena nisu vidljiva u opštem programu. Kaže se da prostor imena predstavlja nezavisan prostor deklaracije ili opseg.

Pomozimo našem zajedničkom prijatelju obezbeđivanjem odvojenih prostora imena za dve instance klase `Point`:

```
namespace DisneyAnimation_2DGraphics
{
    public class Point(...)

    //...
}

namespace DreamWorksAnimation_3DGraphics
{
    public class Point(...)

    //...
}
```

2. Samo se prostori imena i tipovi mogu deklarisati u opštem prostoru imena. Funkcija se može deklarisati samo kao članica klase. Objekat koji predstavlja podatak može biti ili član klase ili lokalni objekat u funkciji kao što je naša deklaracija `greeting`.

Ključna reč `namespace` predstavlja definiciju prostora imena. Ono što sledi iza ove ključne reči je ime koje jedinstveno identifikuje prostor imena. (Ako više puta upotrebimo ime postojećeg prostora imena, kompajler će pretpostaviti da želimo da dodamo još jednu deklaraciju u postojeći prostor imena. Činjenica da dve upotrebe imena prostora imena ne dovode do kolizije nam omogućava da deklaraciju prostora imena proširimo kroz datoteke.)

Sadržaj svakog prostora imena je smešten unutar vitičastih zagrada. Naše dve klase `Point` nisu više vidljive opštem programu; svaka je ugnežđena u odgovarajući prostor imena. Kažemo da je svaka član svog odgovarajućeg prostora imena.

Direktiva `using` je u ovom slučaju više nego rešenje. Ako naš prijatelj u programu otvori oba prostora imena –

```
using DisneyAnimation_2DGraphics;
using DreamWorksAnimation_3DGraphics;
```

nekvalifikovana upotreba identifikatora `Point` će još uvek rezultirati greškom za vreme kompajliranja. Da bismo nedvosmisleno referencirali ovu ili onu klasu `Point`, moramo upotrebiti potpuno kvalifikovano ime – na primer,

```
DreamWorksAnimation_3DGraphics.Point origin;
```

Ako ovo čitamo sdesna nalevo, tu je deklarisano da `origin` bude instanca klase `Point` definisane u prostoru imena `DreamWorksAnimation_3DGraphics`.

Dvoznačnošću unutar i između prostora imena rukuje se različito u zavisnosti od primećene količine kontrola među kojima postoji sukob imena. U najjednostavnijem slučaju, dve upotrebe istog imena se pojavljuju u jednom prostoru deklaracije i kada je druga upotreba imena primećena odmah, rezultiraju greškom za vreme kompajliranja. Pretpostavka je da programer kome se ovo desilo ima mogućnost da izmeni ili dodeli novo ime identifikatoru u radnom prostoru deklaracije gde se sukob imena pojavio.

Manje je jasno šta bi trebalo da se desi ako se sukob pojavi između prostora imena. U jednom slučaju, otvaramo dva nezavisna prostora imena od kojih svaki sadrži upotrebu istog imena, kao dve instance klase `Point`. Ako smo napravili eksplicitnu upotrebu više puta definisanih identifikatora `Point`, generiše se greška; kompajler ne pokušava da prioritet da jednoj upotrebi u odnosu na drugu ili da na neki drugi način reši dvoznačnu referencu. Jedno rešenje, kao što

smo malopre videli, jeste da kvalifikujemo pristup svakog identifikatora. Alternativno, možemo definisati pseudonim za jednu ili sve više puta definisane instance. Ovo možemo uraditi sa varijantom direktive `using` koja sledi:

```
namespace GameApp
{
    // definiše dve instance tipa Point
    using DisneyAnimation_2DGraphics;
    using DreamWorksAnimation_3DGraphics;

    // OK: Naprave se jedinstveni identifikatori
    // za svaku instancu
    using Point2D = DisneyAnimation_2DGraphics.Point;
    using Point3D = DreamWorksAnimation_3DGraphics.Point;

    class myClass
    {
    Point2D thisPoint;
    Point3D thatPoint;
    }
}
```

Pseudonim je ispravan samo u trenutnom prostoru deklaracije. To znači da on ne uvodi dodatno stalno ime tipa povezano sa klasom. Ako pokušamo da ga koristimo među prostorima imena, kao u ovom slučaju:

```
namespace GameEngine
{
    class App
    {
    // greška: nije prepoznato
    private GameApp.Point2D origin;
    }
}
```

kompajler neće ništa uraditi i generisaće sledeću poruku:

```
The type or namespace name 'Point2D' does not exist in the
class or namespace 'GameApp'
```

(Tip ili prostori imena sa imenom 'Point2D' ne postoje u klasi ili prostoru imena 'GameApp')

Kada koristimo prostor imena, uglavnom ne znamo koliko je u njemu definisano imena. Bilo bi vrlo neprijatno ako bismo, svaki put kada dodamo novi pro-

stor imena, morali zadržavati dah dok ponovo kompajliramo kako bismo videli da li će doći do greške. Zbog toga nam jezik minimizuje smetnje koje programu može naneti otvaranje prostora imena.

Ako su dve ili više instanci identifikatora, kao što su naše dve klase `Point`, vidljivi u radnom prostoru deklaracije preko višestrukih direktiva `using`, greška će se pojaviti samo pri nekvalifikovanoj upotrebi identifikatora. Ako ne pristupimo identifikatoru, još uvek može da nastane dvoznačnost, a neće se pojaviti ni greška ni upozorenje.

Ako identifikator, vidljiv preko direktive `using`, duplicira identifikator koji smo mi definisali, naš identifikator ima prednost. Nekvalifikovana upotreba identifikatora se uvek razrešava na instancu koju smo mi definisali. Efekat koji dobijamo je da naša instanca sakriva vidljivost identifikatora u prostoru imena. Program nastavlja da radi isto tako kao što je radio pre upotrebe dodatnog prostora imena.

Kakve vrste imena treba davati prostorima imena? Opšta imena, kao što su `Drawing`, `Data`, `Math` i slično, teško da će biti jedinstvena. Preporučuje se dodavanje prefiksa koji identifikuje organizaciju ili projektnu grupu.

Prostori imena su neophodni elementi komponentnog razvoja. Kao što smo videli, oni omogućavaju ponovnu upotrebu našeg softvera u programskim okruženjima. Ipak, za manje ambiciozne programe, kao što je `Hello` program sa početka ovog poglavlja, korišćenje prostora imena je nije potrebno.

1.3 Alternativni oblici funkcije `Main()`

U preostalom delu ovog poglavlja, dok budemo implementirali mali program `WordCount`, razmotrićemo unapred definisane elemente jezika C#. `WordCount` otvara korisnički definisanu tekstualnu datoteku i izračunava broj pojava svake reči u toj datoteci. Rezultati se sortiraju po rečničkom redosledu i upisuju se u izlaznu datoteku. Pored toga, program podržava dve opcije koje se mogu proslediti iz komandne linije:

1. `-t` dovodi do toga da program uključi praćenje toka; podrazumeva se da je praćenje toka isključeno.
2. `-s` dovodi do toga da program izračuna i obavesti ukupno vreme koje mu je bilo potrebno da pročita datoteku, obradi reči i upiše rezultat; podrazumeva se da se o vremenima ne daju izveštaji.

Naš prvi zadatak u funkciji `Main()` jeste da, ako ih ima, pristupimo parametrima prosleđenim u program iz komandne linije. To možemo uraditi tako što ćemo upotrebiti drugi oblik funkcije Main() koji definiše potpis funkcije sa jednim parametrom:

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

`args` je definisan kao niz stringova. Niz `args` se automatski popunjava bilo kojim argumentima koje korisnik prosledi iz komandne linije. Na primer, ako korisnik pozove naš program na sledeći način:

```
WordCount -s mytext.txt
```

prvi element niza `args` sadržaće element `-s`, a drugi `mytext.txt`.

Takođe, bilo koji oblik funkcije `Main()` može eventualno vratiti vrednost tipa `int`:

```
public static int Main() {}
public static int Main( string [] args ) {}
```

Vraćena vrednost se posmatra kao izlazni status programa. Po dogovoru, vraćena vrednost 0 znači da se program izvršio uspešno. Vrednosti različite od nule označavaju neki oblik greške u programu. Paradoksalno, povratni tip `void` interno rezultira kao povratni status 0; to znači da izvršno okruženje program uvek posmatra kao uspešno izvršen. U sledećem odeljku saznaćemo kako možemo upotrebiti drugi oblik funkcije `Main()`.

1.4 Pravljenje iskaza

Prva stvar koju treba uraditi jeste da odredimo da li je korisnik prosledio neke argumente. Ovo možemo uraditi pitajući niz `args` koliko elemenata sadrži[3]. Odlučio sam da se naš program isključi ako korisnik ne obezbedi potrebne argumente iz komandne linije. (Kao vežbanje, možda želite da program implementirate tako da korisniku omogući interaktivni unos željenih opcija. Na ovaj način program je, naravno, druželjubiviji.)

3. U jeziku C# ne možemo napisati `if(!arts.Length)` kako bismo proverili da li je niz prazan, jer se 0 ne interpretira sa značenjem netačno.

U mojoj implementaciji, ako je niz `args` prazan, program štampa objašnjenje pravog načina pozivanja programa `WordCount`, a onda izlazi pomoću naredbe `return`. (Naredba `return` prouzrokuje da se funkcija, u kojoj se naredba `return` pojavila, prekine – odnosno, da se vrati na mesto sa kojeg je pozvana.)

```
        public static void Main( string [] args )
        {
          if ( args.Length == 0 )
          {
               display_usage();
               return;
          }
    }
```

`Length` je *svojstvo* niza. U njemu se nalazi broj elemenata koji trenutno postoje u nizu. Provera dužine je smeštena unutar uslovne naredbe `if` jezika C#. Ako provera dobije vrednost tačno, izvršava se iskaz koji odmah sledi iza provere; inače, on se ignoriše. Ako treba izvršiti više iskaza, kao u našem primeru, oni moraju biti ograničeni vitičastim zagradama (tekst unutar zagrada se naziva *blok iskaza*).

Uobičajena greška koju početnici prave je da zaborave blok iskaza kada žele da izvrše dva ili više iskaza[4]:

```
// ovo je neispravna upotreba naredbe if
if ( args.Length == 0 )
     display_usage();
     return;
```

Uvlačenje naredbe `return` prikazuje programerovu nameru. Ono ipak ne reflektuje ponašanje programa. Bez bloka iskaza uslovno se izvršava samo funkcija; iskaz `return` se izvršava bez obzira da li je niz prazan ili ne.

Iskaz `return` takođe može da vrati vrednost. Ova vrednost postaje povratna vrednost funkcije – na primer

4. Svi ovi fragmenti koda se javljaju u funkciji `Main()`. Da bih sačuvao prostor, ne prikazujem zatvarajući deo funkcije `Main()`.