Kako da stupite u kontakt s nama

Komentare i pitanja koji se odnose na izvorno izdanje ove knjige šaljite na adresu:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (u SAD ili u Kanadi)
707-829-0515 (međunarodni ili lokalni pristup)
707-829-0104 (faks)

Na veb stranicama ove knjige objavljuju se ispravke, primeri i sve dodatne informacije. Možete joj pristupiti na adresi:

http://oreil.ly/CSharp5_PR

Komentare ili tehnička pitanja o ovoj knjizi, pošaljite na sledeću adresu:

bookquestions@oreilly.com

Više informacija o knjigama, kursevima, konferencijama i novostima kompanije O'Reilly, naći ćete na adresi *http://www.oreilly.com*.

Potražite ih na Facebooku: *http://facebook.com/oreilly*

Pratite ih na Twitteru: *http://twitter.com/oreillymedia*

Gledajte ih na YouTubeu: *http://www.youtube.com/oreillymedia*

Informacije o izdanjima Mikro knjige potražite na adresi:

http://www.mikroknjiga.rs

Prvi program na jeziku C#

Evo programa koji množi 12 sa 30 i ispisuje rezultat, 360, na ekranu. Dupla kosa crta ukazuje na to da je ostatak reda *komentar*.

```
using System;                   // Učitavanje imenskog prostora
class Test                      // Deklaracija klase
{

   static void Main()           // Deklaracija metode
   {
```

```
        int x = 12 * 30;          // Naredba 1
Console.WriteLine (x);            // Naredba 2
 }                                // Kraj metode
 }                                // Kraj klase
```

U srcu ovog programa su dve naredbe (engl. *statements*). Naredbe se u jeziku C# izvršavaju sekvencijalno i završavaju znakom tačka i zarez. Prva naredba izračunava *izraz* `12 * 30` i rezultat smešta u *lokalnu promenljivu* po imenu `x`, celobrojnog tipa (engl. *integer*). Druga naredba poziva *metodu* `WriteLine` klase `Console` da ispiše promenljivu `x` u tekstualnom prozoru na ekranu.

Metoda izvršava akciju u obliku niza naredaba koji se naziva *blok naredaba* (engl. *statement block*) – par vitičastih zagrada koje sadrže nula ili više naredaba. Definisali smo samo jednu metodu, po imenu `Main`.

Pisanje funkcija višeg nivoa koje pozivaju funkcije nižeg nivoa pojednostavljuje program. Evo kako ćemo *refaktorisati* program pomoću metode koja se može ponovo upotrebiti i koja množi ceo broj sa 12:

```
using System;

class Test
{
  static void Main()
  {
    Console.WriteLine (FeetToInches (30)); // 360
    Console.WriteLine (FeetToInches (100)); // 1200
 }

  static int FeetToInches (int feet)
  {
    int inches = feet * 12;
    return inches;
  }
}
```

Metoda može da primi *ulazne podatke* (engl. *input*) od pozivaoca tako što joj se zadaju parametri i može da vrati *rezultat* (engl. *output*) pozivaocu tako što joj se zada *povratni tip* (engl. *return type*). Definisali smo

metodu `FeetToInches` koja ima parametar za unošenje stopa (`feet`), i povratni tip za rezultat u inčima, pri čemu su oba tipa `int` (celi brojevi).

Literali `30` i `100` su *argumenti* koji su prosleđeni metodi `FeetToInches`. U našem primeru, metoda `Main` ima prazne zagrade zato što nema parametre, i tipa je `void` zato što svom pozivaocu ne vraća nikakvu vrednost. C# prepoznaje metodu po imenu `Main` kao mesto podrazumevane ulazne tačke izvršavanja. Metoda `Main` može opciono da vrati tip `int` (umesto `void`) kako bi vratila neku vrednost izvršnom okruženju. Uz to, metoda `Main` može opciono da prihvati niz znakovnih nizova (stringova) kao parametar (niz će sačinjavati argumenti prosleđeni izvršnom programu). Na primer:

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

NAPOMENA

Niz (kao što je `string[]`) predstavlja fiksan broj elemenata određenog tipa (više informacija naći ćete u odeljku „Nizovi“, na strani 33).

Metode su jedna od nekoliko vrsta funkcija u jeziku C#. Koristili smo i drugu vrstu funkcija, *operator* `*`, koji služi za množenje. Osim toga, postoje još i *konstruktori*, *svojstva* (engl. *properties*), *događaji* (engl. *events*), *indekseri* i *finalizatori*.

U našem primeru, navedene dve metode grupisane su u klasu. *Klasa* grupiše funkcije članice i podatke članove da bi se formirao objektno orijentisan gradivni blok. Klasa `Console` grupiše članove zadužene za funkcionisanje ulaza/izlaza s komandne linije, kao što je metoda `WriteLine`. Naša klasa `Test` grupiše dve metode – `Main` i `FeetToInches`. Klasa je neka vrsta *tipa*, o čemu će biti reči u odeljku „Osnove tipova“ na strani 11.

Na krajnjem spoljnom nivou svakog programa, tipovi su organizovani u *imenske prostore* (engl. *namespaces*). Direktiva `using` je zadata da bi imenski prostor `System` bio dostupan našoj aplikaciji, kako bi kori-

stila klasu `Console`. Sve klase unutar imenskog prostora `TestPrograms` možemo definisati na sledeći način:

```
using System;

namespace TestPrograms
{

  class Test {...}
  class Test2 {...}
}
```

.NET Framework je organizovan u obliku ugnežđenih (engl. *nested*) imenskih prostora. Na primer, evo imenskog prostora koji sadrži tipove za rad sa tekstom:

```
using System.Text;
```

Direktiva `using` je tu radi pogodnosti; tip možete referencirati i navođenjem njegovog potpuno kvalifikovanog imena, tj. imena tipa kojem prethodi njegov imenski prostor – na primer, `System.Text.String Builder`.

Prevođenje programa

C# kompajler (prevodilac programa) pakuje izvorni kôd, zadat kao skup datoteka s nastavkom *.cs*, u *sklop* (engl. *assembly*). Sklop je jedinica pakovanja i primene u okruženju .NET. Sklop može biti ili *aplikacija* ili *biblioteka*. Standardna konzolna ili Windows aplikacija jeste datoteka tipa *.exe* i ima metodu `Main`. Biblioteka je datoteka tipa *.dll* i ekvivalentna je *.exe* datoteci, s tim što nema ulaznu tačku. Nju poziva (*referencira*) neka aplikacija ili druge biblioteke. .NET Framework je skup biblioteka.

C# kompajler se zove *csc.exe*. Za kompajliranje (prevođenje) možete koristiti neko integrisano razvojno okruženje (IDE), kao što je Visual Studio, ili ručno pozvati kompajler `csc` s komandne linije. Da biste program preveli ručno, prvo ga snimite u datoteku, npr. *MyFirstProgram.cs*, a zatim idite na komandnu liniju i pozovite `csc` (smešten u *%SystemRoot%\Microsoft.NET\Framework\<framework-version>* gde je *%SystemRoot%* vaš Windows direktorijum) na sledeći način:

```
csc MyFirstProgram.cs
```

Dobija se aplikacija pod imenom *MyFirstProgram.exe*. Da biste napravili biblioteku (*.dll*), uradite sledeće:

```
csc /target:library MyFirstProgram.cs
```

Sintaksa

Sintaksa jezika C# inspirisana je sintaksom programskih jezika C i C++. U ovom odeljku opisaćemo elemente sintakse jezika C# na primeru sledećeg programa:

```
using System;
class Test
{
 static void Main()
 {
 int x = 12 * 30;
 Console.WriteLine (x);
 }
}
```

Identifikatori i rezervisane reči

Identifikatori su imena koja programeri biraju za svoje klase, metode, promenljive itd. U našem primeru programa, identifikatori su:

```
System   Test   Main   x   Console   WriteLine
```

Identifikator mora biti cela reč, sastavljena isključivo od Unicode znakova, i mora počinjati slovom ili znakom za podvlačenje. U C# identifikatorima pravi se razlika između malih i velikih slova. Po konvenciji, parametri, lokalne promenljive i privatna polja pišu se tzv. kamiljom notacijom – CamelCase – (npr., `mojaPromenljiva`), a svi drugi identifikatori – Pascal notacijom (npr., `MojaMetoda`).

Rezervisane reči (engl. *keywords*) jesu imena koja je rezervisao kompajler i koja ne možete koristiti kao identifikatore. U našem primeru, to su sledeće reči:

```
using   class   static   void   int
```

Evo i kompletne liste rezervisanih reči u jeziku C#:

```
abstract      enum          long          stackalloc
as            event         namespace     static
base          explicit      new           string
bool          extern        null          struct
break         false         object        switch
byte          finally       operator      this
case          fixed         out           throw
catch         float         override      true
char          for           params        try
checked       foreach       private       typeof
class         goto          protected     uint
const         if            public        ulong
continue      implicit      readonly      unchecked
decimal       in            ref           unsafe
default       int           return        ushort
delegate      interface     sbyte         using
do            internal      sealed        virtual
double        is            short         void
else          lock          sizeof        while
```

Izbegavanje konflikata

Ako ipak želite da koristite rezervisanu reč kao identifikator, morate ispred nje staviti znak `@`. Na primer:

```
class class {...}   // Neispravno
class @class {...}  // Ispravno
```

Simbol `@` nije deo identifikatora. Znači, `@mojaPromenljiva` je isto što i `mojaPromenljiva`.

Kontekstualne rezervisane reči

Neke rezervisane reči su *kontekstualne*, što znači da se mogu koristiti i kao identifikatori – bez znaka `@`. To su:

```
add          equals   join      set
ascending    from     let       value
async        get      on        var
await        global   orderby   where
by           group    partial   yield
descending   in       remove
dynamic      into     select
```

Kontekstualne rezervisane reči ne mogu biti dvosmislene u kontekstu u kome su upotrebljene.

Literali, znakovi interpunkcije i operatori

Literali su prosti delovi podataka, leksički ugrađeni u program. U našem primeru, literali su `12` i `30`. *Znakovi interpunkcije* pomažu da se označi struktura programa. U našem programu, to su {, } i `;`.

Vitičaste zagrade grupišu više naredaba u blok naredaba. Znak tačka i zarez završava (pojedinačnu) naredbu. Naredbe se mogu prelamati u više redova:

```
Console.WriteLine
  (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10);
```

Operator transformiše i kombinuje izraze. Većina operatora u jeziku C# pišu se pomoću simbola; takav je, na primer, i operator množenja, `*`. U našem programu, operatori su:

```
. () * =
```

Tačka označava pristupanje članu (ili decimalnu tačku u numeričkim literalima). Zagrade se koriste pri deklarisanju ili pozivanju metode; prazne zagrade se koriste kada metoda ne prihvata nijedan argument.

Znak jednakosti obavlja operaciju *dodele* (engl. *assignment*), dok dvostruki znak jednakosti, ==, poredi dve vrednosti kako bi se utvrdilo da li su jednake.

Komentari

C# podržava dva stila dokumentovanja izvornog koda: *jednoredne komentare* i *višeredne komentare*. Jednoredni komentar počinje sa dve kose crte i nastavlja se do kraja reda. Na primer:

```
int x = 3; // Komentar o dodeljivanju vrednosti 3 promenljivoj x
```

Višeredni komentar počinje znakovima /* a završava znakovima */. Na primer:

```
int x = 3; /* ovo je komentar koji se proteže
u dva reda */
```

Komentari mogu da sadrže XML dokumentacione oznake (videti „XML dokumentacija“ na strani 202).

Osnove tipova

Tip definiše prirodu date vrednosti. U našem primeru koristili smo dva literala tipa `int` s vrednostima 12 i 30. Uz to, deklarisali smo i *promenljivu* tipa `int` po imenu `x`.

Promenljiva (engl. *variable*) označava lokaciju u memoriji koja može da sadrži različite vrednosti u različito vreme. Nasuprot tome, *konstanta* uvek predstavlja istu vrednost (više o tome kasnije).

U jeziku C#, svaka vrednost je *instanca* određenog tipa. Značenje same vrednosti i skup mogućih vrednosti date promenljive, određeni su njenim tipom.

Primeri unapred definisanih tipova

Unapred definisani tipovi, engl. *predefined types* (zovu se i ugrađeni tipovi, engl. *built-in types*) jesu tipovi koje kompajler standardno podržava. Tip `int` je unapred definisan tip namenjen za predstavljanje

skupa celih brojeva (engl. *integers*) koji staju u 32 bita memorije, od -2^{31} do $2^{31}-1$. Evo kako možemo da izvršavamo aritmetičke funkcije sa instancama tipa `int`:

```
int x = 12 * 30;
```

Još jedan od unapred definisanih tipova u jeziku C# jeste `string`. Tip `string` predstavlja znakovne nizove, tj. sekvence znakova – na primer, „.NET“ ili „*http://oreilly.com*“. Sa znakovnim nizovima možemo raditi tako što ih pozivamo pomoću funkcija:

```
string message = "Hello world";
string upperMessage = message.ToUpper();
Console.WriteLine (upperMessage);        // HELLO WORLD

int x = 2014;
message = message + x.ToString();
Console.WriteLine (message);             // Hello world2014
```

Unapred definisan tip `bool` ima samo dve moguće vrednosti: `true` i `false`. Tip `bool` se obično koristi za uslovno grananje izvršnog toka programa s naredbom `if`. Na primer:

```
bool simpleVar = false;
if (simpleVar)
  Console.WriteLine ("This will not print");

int x = 5000;
bool lessThanAMile = x < 5280;
if (lessThanAMile)
  Console.WriteLine ("This will print");
```

Napomena

Imenski prostor `System` u .NET Frameworku sadrži mnoge važne tipove koji u jeziku C# nisu unapred definisani (npr., `DateTime`).

Primeri namenskih tipova

Kao što složene funkcije možemo da gradimo od onih jednostavnih, tako i složene tipove možemo praviti od osnovnih. U ovom primeru definisaćemo namenski tip (engl. *custom type*) po imenu `UnitConverter` – klasu koja služi za konverziju mernih jedinica:

```
using System;
public class UnitConverter
{
  int ratio;                                  // Polje

  public UnitConverter (int unitRatio)        // Konstruktor
  {
    ratio = unitRatio;
  }

  public int Convert (int unit)               // Metoda
  {
    return unit * ratio;
  }
}

class Test
{
  static void Main()
 {
    UnitConverter feetToInches = new UnitConverter(12);
    UnitConverter milesToFeet = new UnitConverter(5280);

    Console.Write (feetToInches.Convert(30));   // 360
    Console.Write (feetToInches.Convert(100));  // 1200
    Console.Write (feetToInches.Convert
                    (milesToFeet.Convert(1)));  // 63360
  }
}
```

Članovi tipa

Tip sadrži *podatke članove* (engl. *data members*) i *funkcije članice* (engl. *function members*). Podatak član namenskog tipa `UnitConverter` jeste *polje* (engl. *field*) po imenu `ratio`. Funkcije članice tipa `UnitConverter` jesu metoda `Convert` i konstruktor `UnitConvertera`.

Simetrija između unapred definisanih tipova i namenskih tipova

U jeziku C# zgodno je to što se unapred definisani tipovi i namenski tipovi ne razlikuju mnogo. Unapred definisan tip `int` služi za cele brojeve. On sadrži podatke – 32 bita – i stavlja ih na raspolaganje funkcijama članicama, npr. funkciji `ToString`. Slično tome, naš namenski tip `UnitConverter` služi za konverzije mernih jedinica. On sadrži podatke – u ovom slučaju, `ratio` – i stavlja ih na raspolaganje funkcijama članicama koje ih koriste.

Konstruktori i instanciranje

Podaci nastaju *instanciranjem* tipa. Unapred definisani tipovi instanciraju se jednostavno – korišćenjem literala kao što je `12` ili `"Hello, world"`.

Instance namenskog tipa pravi operator `new`. Metodu `Main` smo započeli tako što smo napravili dve instance tipa `UnitConverter`. Odmah nakon što operator `new` instancira objekat, poziva se *konstruktor* tog objekta da ga inicijalizuje. Konstruktor se definiše kao metoda, s tim što su ime metode i tip rezultata (povratne vrednosti) ime samog tipa:

```
public UnitConverter (int unitRatio)  // Konstruktor
{
 ratio = unitRatio;
}
```

Poređenje članova instanci i statičkih članova

Podaci članovi i funkcije članice koji deluju na *instancu* datog tipa zovu se članovi instance (engl. *instance members*). Metoda `Convert` tipa `UnitConverter` i metoda `ToString` tipa `int` primeri su članova instance. Članovi tipa su podrazumevano članovi instance.

Podaci članovi i funkcije članice koji ne deluju na instancu tipa nego na sam tip, moraju biti označeni kao `static`. Metode `Test.Main` i `Console.WriteLine` jesu statičke metode. Klasa `Console` je, u stvari, *statička klasa*, što znači da su *svi* njeni članovi statički. Nikad se ne prave instance klase `Console` – jednu konzolu deli cela aplikacija.

Da bismo istakli razlike između članova instance i statičkih članova, reći ćemo da se polje `Name` instance odnosi na jednu određenu instancu klase `Panda`, dok se `Population` odnosi na skup svih instanci klase `Panda`:

```
public class Panda
{
  public string Name;           // Polje instance
  public static int Population; // Statičko polje

  public Panda (string n)       // Konstruktor
  {
    Name = n;                   // Dodeljivanje vrednosti polju
                                // instance
     Population = Population+1; // Inkrementiranje statičkog
                                // polja
  }
}
```

Naredni kôd pravi dve instance klase `Panda`, ispisuje njihova imena, a zatim ispisuje ukupan broj instanci klase:

```
Panda p1 = new Panda ("Pan Dee");
Panda p2 = new Panda ("Pan Dah");

Console.WriteLine (p1.Name);          // Pan Dee
Console.WriteLine (p2.Name);          // Pan Dah
Console.WriteLine (Panda.Population); // 2
```

Rezervisana reč public

Rezervisana reč `public` izlaže članove klase drugim klasama. U ovom primeru, kada polje `Name` klase `Panda` ne bi bilo javno, klasa `Test` ne bi mogla da mu pristupi. Označavanjem svog člana rezervisanom rečju

`public`, tip kaže: „Evo šta želim da vide ostali tipovi – sve ostalo su moji privatni detalji implementacije.“ U terminologiji objektno orijentisanog programiranja, kažemo da javni članovi *kapsuliraju* (engl. *encapsulate*) privatne članove klase.

Konverzije

C# može da konvertuje instance kompatibilnih tipova jedne u druge. Konverzijom uvek nastaje nova vrednost od postojeće. Konverzije mogu biti *implicitne* ili *eksplicitne*: implicitne konverzije se obavljaju automatski, dok je za eksplicitne konverzije potrebno *pretvaranje* (engl. *cast*). U sledećem primeru, *implicitno* konvertujemo tip `int` u tip `long` (koji ima dvaput veći kapacitet u bitovima od `int`) i *eksplicitno* konvertujemo tip `int` u tip `short` (koji ima upola manji kapacitet u bitovima od `int`):

```
int x = 12345;       // int je 32-bitni ceo broj
long y = x;          // Implicitna konverzija u 64-bitni ceo
                     // broj
short z = (short)x;  // Eksplicitna konverzija u 16-bitni
                     // ceo broj
```

U opštem slučaju, implicitne konverzije su dozvoljene kada kompajler može da garantuje da će one uvek uspeti bez gubljenja informacija. U suprotnom, morate obaviti eksplicitnu konverziju između kompatibilnih tipova.

Poređenje vrednosnih i referentnih tipova

U jeziku C#, tipovi se mogu podeliti na *vrednosne* (engl. *value types*) i *referentne* (engl. *reference types*).

U *vrednosne tipove* spada većina ugrađenih tipova (konkretno, svi numerički tipovi, tip `char` i tip `bool`) kao i namenski tipovi `struct` i `enum`. U *referentne tipove* spadaju svi tipovi klasa, nizova, delegata i interfejsa.

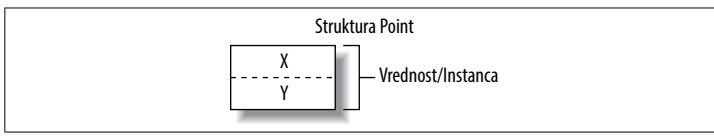
Osnovna razlika između vrednosnih i referentnih tipova jeste način rada s njima u memoriji.

Vrednosni tipovi

Sadržaj pomenljive ili konstante *vrednosnog tipa* jeste samo neka vrednost. Na primer, ugrađen vrednosni tip `int` sadrži 32 bita podataka.

Namenski vrednosni tip možete definisati pomoću rezervisane reči `struct` (slika 1):

```
public struct Point { public int X, Y; }
```



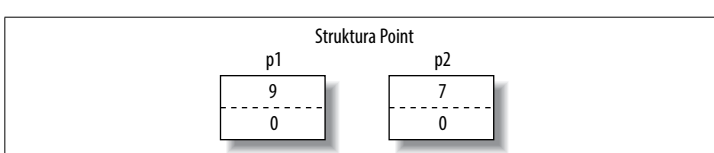
Slika 1. Instanca vrednosnog tipa u memoriji

Kada se instanca vrednosnog tipa dodeljuje drugoj instanci, postojeća instanca se uvek kopira. Na primer:

```
Point p1 = new Point();
p1.X = 7;

Point p2 = p1;             // Dodeljivanje uzrokuje kopiranje
Console.WriteLine (p1.X);  // 7
Console.WriteLine (p2.X);  // 7
p1.X = 9; // Promena p1.X
Console.WriteLine (p1.X);  // 9
Console.WriteLine (p2.X);  // 7
```

Slika 2 prikazuje da su `p1` i `p2` uskladišteni nezavisno.

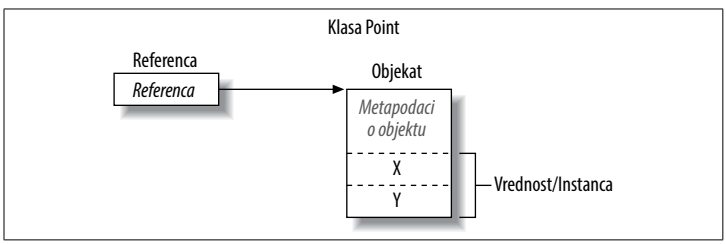


Slika 2. Pri dodeljivanju se instanca vrednosnog tipa kopira

Referentni tipovi

Referentni tip je složeniji od vrednosnog, i ima dva dela: *objekat* i *referencu* na taj objekat. Promenljiva ili konstanta referentnog tipa sadrži referencu na objekat u kome se nalazi vrednost. Evo tipa `Point` iz prethodnog primera, s tim što je sada napisan kao klasa (slika 3):

```
public class Point { public int X, Y; }
```



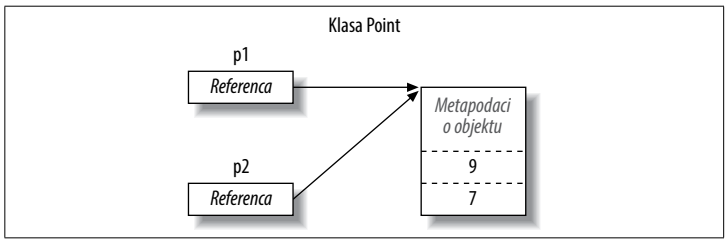
Slika 3. Instanca referentnog tipa u memoriji

Kada se instanca referentnog tipa dodeljuje drugoj instanci, kopira se referenca instance a ne objekat. To omogućava da više promenljivih referencira isti objekat – što obično nije moguće postići s vrednosnim tipovima. Ako ponovimo prethodni primer, ali s promenljivom `Point` koja je sada klasa, operacija nad `p1` utiče i na `p2`:

```
Point p1 = new Point();
p1.X = 7;

Point p2 = p1;             // Kopira referencu na p1
Console.WriteLine (p1.X);  // 7
Console.WriteLine (p2.X);  // 7
p1.X = 9;                  // Promena p1.X
Console.WriteLine (p1.X);  // 9
Console.WriteLine (p2.X);  // 9
```

Slika 4 prikazuje da su `p1` i `p2` dve reference koje upućuju na isti objekat.



Slika 4. Dodeljivanjem se kopira referenca

Vrednost null

Referentnom tipu se može dodeliti literal `null`, što znači da ta referenca ne upućuje ni na jedan objekat. Pod pretpostavkom da je `Point` klasa:

```
Point p = null;
Console.WriteLine (p == null); // true
```

Pristupanje članu s referencom `null`, generiše grešku pri izvršavanju (engl. *runtime error*):

```
Console.WriteLine (p.X); // NullReferenceException
```

Suprotno tome, vrednosni tip obično ne može da ima vrednost `null`:

```
struct Point {...}
...
Point p = null; // Greška pri prevođenju
int x = null;   // Greška pri prevođenju
```

NAPOMENA

C# ima specijalan konstrukt pod imenom *tip koji prihvata vrednost null* (engl. *nullable type*) za predstavljanje praznih elemenata vrednosnog tipa (više informacija u odeljku „Tipovi koji prihvataju null“ na strani 135).
