2

Osnove jezika C#

U ovom poglavlju predstavljamo osnove jezika C#.

Svi programi i blokovi koda iz ovog i dva naredna poglavlja na raspolaganju su u obliku interaktivnih primera u LINQPadu. Proučavanje tih primera u kombinaciji s ovom knjigom ubrzava učenje jer možete menjati primere i odmah videti rezultat bez izrade projekata i rešenja u Visual Studiju.

Da biste preuzeli primere, mišem pritisnite jezičak u LINQPadu a zatim izaberite „Download more samples.“ LINQPad je besplatan – idite na adresu *http://www.linqpad.net*.

Prvi C# program

Sledeći program množi 12 sa 30 i prikazuje na ekranu rezultat 360. Dvostruka kosa crta znači da je preostali deo reda *komentar*.

```
using System;                          // Uvoženje 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
```

Srce ovog programa čine dve *naredbe* (engl. *statements*):

```
int x = 12 * 30;
Console.WriteLine (x);
```

Naredbe se u jeziku C# izvršavaju jedna za drugom, a završavaju se znakom tačka i zarez (ili *blokom koda*, kao što ćemo videti u nastavku). Prva naredba izračunava vrednost *izraza* `12 * 30` i rezultat smešta u *lokalnu promenljivu*, čije je ime `x`, a tip celobrojni (engl. *integer*). Druga naredba poziva *metodu* `WriteLine` klase `Console`, koja prikazuje vrednost promenljive `x` u obliku teksta na ekranu.

Metoda izvršava određenu akciju pomoću grupe naredaba, koja se zove *blok naredaba* (engl. *statement block*) – što je par vitičastih zagrada između kojih se nalazi nula, jedna ili više naredaba. Definisali smo samo jednu metodu, po imenu `Main`:

```
static void Main()
{
  ...
}
```

Pisanje funkcija višeg nivoa koje pozivaju druge funkcije nižeg nivoa pojednostavljuje program. Evo kako ćemo *refaktorisati* (tj. poboljšati) program pomoću višekratno upotrebljive metode koja množi zadatu celobrojnu vrednost 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 prima *ulazne podatke* (engl. *input*) od svog pozivaoca tako što definiše *parametre*, a može i da mu vraća *izlazne podatke* (engl. *output*) ako definiše *povratni tip* (engl. *return type*). Definisali smo metodu nazvanu `FeetToInches` (pretvaranje stopa u inče) koja ima ulazni parametar za stope i povratni tip za izračunate inče:

```
static int FeetToInches (int feet ) {...}
```

Literali `30` i `100` su *argumenti* koje pozivalac prosleđuje metodi `FeetToInches`. Metoda `Main` iz našeg primera ima prazne zagrade iza imena zato što nema ulazne parametre, a njen povratni tip je `void` jer svom pozivaocu ne vraća nikakvu vrednost:

```
static void Main()
```

Metodu čije je ime `Main` C# prepoznaje kao podrazumevanu ulaznu tačku u programu, odakle počinje izvršavanje programa. Metoda `Main` može po potrebi imati povratnu vrednost celobrojnog tipa (umesto `void`), koju vraća izvršnom okruženju (gde je uobičajeno da vrednost različita od nule znači grešku). Metoda `Main` može opciono da prihvata kao ulazni parametar niz znakovnh vrednosti.

Na primer:

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

Niz (kao što je `string[]`) predstavlja fiksan broj elemenata određenog tipa. Nizove definišete tako što napišete uglaste zagrade iza tipa elemenata niza. Nizovi su opisani u odeljku „Nizovi“ na strani 38.

Metode su jedna od više vrsta funkcija u jeziku C#. Druga vrsta funkcije koju smo pozvali u našm primeru progama jeste *operator* *, koji obavlja množenje. Postoje još i *konstruktori*, *svojstva* (engl. *properties*), *događaji* (engl. *events*), *indekseri* i *finalizatori*.

U našem primeru, navedene dve metode su grupisane u jednu klasu. *Klasa* grupiše funkcije članice i podatke članove tako da svi zajedno čine celinu i objektno orijentisan gradivni blok jezika. Klasa `Console` grupiše članove koji obezbeđuju funkcionalnost učitavanja po-

dataka sa komandne linije i prikazivanje podataka na njoj, kao što je metoda `WriteLine`. Naša klasa `Test` grupiše dve metode – metodu `Main` i metodu `FeetToInches`. Klasa je jedna vrsta *tipa*, što ćemo razmatrati u odeljku „Osnove tipova".

Na najvišem nivou programa, tipovi su organizovani u *imenske prostore* (engl. *namespaces*). Direktiva `using` uključuje imenski prostor `System` u našu aplikaciju, kako bismo mogli da koristimo klasu `Console`. Sve svoje klase možemo da definišemo unutar imenskog prostora `TestPrograms`, na sledeći način:

```
using System;

namespace TestPrograms
{
  class Test  {...}
  class Test2 {...}
}
```

.NET Framework je organizovan tako da su njegovi imenski prostori ugnežđeni jedan u drugi. Na primer, sledeći imenski prostor sadrži tipove za rad s tekstom:

```
using System.Text;
```

Upotreba direktive `using` samo je pogodniji oblik za programera; svaki tip možete referencirati i pomoću njegovog potpuno kvalifikovanog imena, koje se sastoji od imena tipa, kojem prethodi ime imenskog prostora (kojem tip pripada). Na primer: `System.Text.StringBuilder`.

Prevođenje programskog koda

Kompajler jezika C# prevodi izvorni kôd, raspodeljen u grupu datoteka s nastavkom *.cs*, u oblik koji se zove *sklop* (engl. *assembly*). Sklop je jedinica pakovanja i distribuiranja .NET softvera. Sklop može biti *aplikacija* ili *biblioteka*. Standardna konzolska ili Windows aplikacija ima metodu `Main` i pakuje se u *.exe* datoteku. Biblioteka ima nastavak *.dll* i ekvivalentna je *.exe* aplikaciji, ali nema ulaznu tačku. Njena svrha je da je druge aplikacije ili biblioteke pozivaju (*referenciraju*). Sam .NET Frejmvork je skup biblioteka.

Ime kompajlera C# koda je *csc.exe*. Da biste prevodili svoje programe, možete ga pozivati unutar nekog integrisanog razvojnog okruženja, kao što je Visual Studio, ili pozovite `csc` ručno, s komandne linije. (Kompajler je na raspolaganju i u obliku biblioteke; vidi poglavlje 27.) Da biste program preveli ručno, prvo ga snimite u datoteku kao što je *MyFirstProgram.cs*, a zatim pređite na komandnu liniju i pokrenite `csc` (nalazi se u *%ProgramFiles(X86)%\msbuild\14.0\bin*), na sledeći način:

```
csc MyFirstProgram.cs
```

Rezultat je aplikacija čije je ime *MyFirstProgram.exe*.

Čudno je da se .NET Framework 4.6 i 4.7 i dalje isporučuju s kompajlerom za C# 5. Da biste dobili pristup sa komandne linije kompajleru za C# 7, morate instalirati Visual Studio 2017 ili MSBuild 15.

Da biste napravili biblioteku (*.dll*), uradite sledeće:

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

Sklopove detaljno objašnjavamo u poglavlju 18.

Sintaksa

Za sintaksu jezika C# uzor je bila sintaksa jezika C i C++. U ovom odeljku opisaćemo elemente sintakse jezika C# pomoću sledećeg primera programa:

```
using System;

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

Identifikatori i rezervisane reči

Identifikatori su imena koje programeri izaberu za svoje klase, metode, promenljive itd. U našem primeru programa imamo sledeće identifikatore, redosledom kojim se pojavljuju:

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

Identifikator mora da se sastoji od jedne reči, uglavnom od Unicode znakova, a počinje slovom ili znakom za podvlačenje. C# pravi razliku između malih i velikih slova u identifikatorima. Uobičajena je konvencija prema kojoj se imena parametara, lokalnih promenljivih i privatnih polja pišu s malim početnim slovom i mešanim ostalim slovima (npr., `ovoJePromenljiva`), a sve ostale identifikatore treba pisati u tzv. Pascal formatu (npr., `OvoJeMetoda`), s velikim početnim slovom.

Rezervisane reči (engl. *keywords*) jesu imena koja imaju posebno značenje za kompajler. U našem primeru programa, imamo sledeće rezervisane reči:

```
using   class   static   void   int
```

Većina rezervisanih reči je zauzeta, pa ih zbog toga ne možete koristiti kao identifikatore u svojim programima. Evo i kompletnog spiska rezervisanih reči jezika C#:

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

Izbegavanje nedoumica

Ako zaista želite da koristite identifikator koji je isti kao jedna od rezervisanih reči, to možete uraditi tako što mu pridružite prefiks `@`. Na primer:

```
class class  {...}      // Nije dozvoljeno
class @class {...}      // Ispravno
```

Pošto simbol `@` nije sastavni deo samog identifikatora, `@imePromenljive` je isto što i `imePromenljive`.

Prefiks `@` može biti koristan kada radite s bibliotekama napisanim na drugim .NET jezicima koji imaju druge rezervisane reči.

Kontekstne rezervisane reči

Neke rezervisane reči su *kontekstne*, što znači da se mogu koristiti kao identifikatori – bez prefiksa `@`. To su sledeće:

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

Pri upotrebi kontekstnih rezervisanih reči, ne može se pojaviti dvosmislenost u kontekstu u kojem se one koriste.

Literali, graničnici i operatori

Literali su jedinice podataka umetnute u program. U našem primeru programa, literali su `12` i `30`.

Graničnici služe za razgraničavanje pojedinih delova strukture programa. U našem primeru programa imamo sledeće graničnike:

```
{   }   ;
```

Vitičaste zagrade grupišu više naredaba u jedan *blok naredaba*.

Svaka naredba se završava znakom tačka i zarez. (Međutim, taj znak nije obavezan iza blokova naredaba.) Jedna naredba može se prostirati na više redova:

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

Operator transformiše i kombinuje delove izraza. Većina operatora jezika C# piše se u obliku simbola, kao što je operator množenja, `*`. Operatore razmatramo detaljnije u nastavku ovog poglavlja. U našem primeru programa, koristimo sledeće operatore:

```
.  ()   *   =
```

Tačkom se obeležava član nečega (ili je decimalna tačka u numeričkim literalima). Zagrade se koriste u pozivima i deklaracijama metoda; prazne zagrade se zadaju kada metoda ne prihvata nijedan argument. (Zagrade imaju i druge namene koje ćemo razmotriti u nastavku vog poglavlja) Znak jednakosti obavlja operaciju *dodeljivanja vrednosti*. (Dvostruki znak jednakosti, `==`, ispituje jednakost, kao što ćemo videti u nastavku ovog poglavlja.)

Komentari

U jeziku C# postoje dve vrste komentara za dokumentovanje izvornog koda: *jednoredni komentari* i *višeredni komentari*. Jednoredni komentar počinje dvostrukom kosom crtom i nastavlja se do kraja reda. Na primer:

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

Višeredni komentar počinje znacima /* i završava se sa */. Na primer:

```
int x = 3;   /* Ovaj komentar se prostire
                u dva reda */
```

U komentare se mogu umetati i dokumentacione XML oznake, što je objašnjeno u odeljku „Dokumentovanje programa u formatu XML“ na strani 182 u poglavlju 4.

Osnove tipova

Tip (engl. *type*) definiše prirodu date vrednosti. U našem primeru, koristimo dva literala tipa `int` čije su vrednosti 12 i 30. Deklarisali smo i *promenljivu* tipa `int` čije je ime `x`:

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

Promenljiva (engl. *variable*) opisuje mesto u memoriji gde se nalaze vrednosti koje se mogu menjati tokom vremena. Za razliku od toga, *konstanta* uvek predstavlja istu vrednost (više o tome u nastavku ovog poglavlja):

```
const int y = 360;
```

Sve vrednosti u C# programu jesu *instance* određenog tipa. Tip promenljive određuje značenje njene vrednosti, kao i skup vrednosti koje promenljiva može imati.

Primeri unapred definisanih tipova

Unapred definisani (standardni, ugrađeni) tipovi jesu tipovi koje kompajler koda posebno podržava. Tip `int` je unapred definisan tip za celobrojne vrednosti koje se mogu smestiti u 32 bita memorije, u opsegu od -2^{31} do $2^{31}-1$. To je podrazumevani tip numeričkih literala unutar tog opsega. Nad instancama tipa `int` možemo obavljati operacije kao što su aritmetičke, na sledeći način:

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

Drugi unapred definisan tip u jeziku C# jeste `string`. Tip `string` predstavlja niz alfanumeričkih znakova, kao što je „.NET“ ili „*http://www.mikroknjiga.rs*“. Znakovne nizove možemo obrađivati pomoću funkcija kao što su sledeće:

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

int x = 2015;
message = message + x.ToString();
Console.WriteLine (message);                  // Hello world2015
```

Unapred definisan tip `bool` može imati jednu od samo dve moguće vrednosti: `true` ili `false`. Tip `bool` se uglavnom koristi za uslovno usmeravanje toka izvršavanja programa pomoću naredbe `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");
```


Jezik C# prepoznaje unapred definisane tipove (koji se nazivaju i ugrađeni tipovi) na osnovu odgovarajućih rezervisanih reči jezika. Imenski prostor `System` .NET Frameworka sadrži brojne važne tipove koji nisu unapred definisani u jeziku C# (na primer, `DateTime`).

Primeri namenskih tipova

Isto kao što možemo graditi složene funkcije kombinujući jednostavne, možemo graditi i složene tipove kombinujući osnovne tipove. U narednom primeru, definišemo namenski (nestandardni) tip (engl. *custom type*) čije je ime `UnitConverter` – klasu koja omogućava konverziju mernih jedinica:

```
using System;

public class UnitConverter
{
  int ratio;                                                  // Polje
  public UnitConverter (int unitRatio) {ratio = unitRatio; } // Konstruktor
  public int Convert   (int unit)    {return unit * ratio; } // Metoda
}

class Test
{
  static void Main()
  {
    UnitConverter feetToInchesConverter = new UnitConverter (12);
    UnitConverter milesToFeetConverter  = new UnitConverter (5280);

    Console.WriteLine (feetToInchesConverter.Convert(30));    // 360
    Console.WriteLine (feetToInchesConverter.Convert(100));   // 1200
    Console.WriteLine (feetToInchesConverter.Convert(
                          milesToFeetConverter.Convert(1)));   // 63360
  }
}
```

Članovi tipa

Svaki tip sadrži *podatke članove* (engl. *data members*) i *funkcije članice* (engl. *function members*). Podatak član klase `UnitConverter` jeste *polje* (engl. *field*) čije je ime `ratio` (faktor konverzije). Funkcije članice klase `UnitConverter` jesu metoda `Convert` i *konstruktor* klase `UnitConverter`.

Simetrija unapred definisanih i namenskih tipova

Jedna od lepota jezika C# jeste činjenica da ima malo razlika između unapred definisanih i namenskih tipova koje sami definišete. Unapred definisan tip `int` opisuje celobrojne vrednosti. On čuva podatke – u 32 bita – i ima funkcije članice za rad s tim podacima, kao što

je metoda `ToString`. Slično tome, naš namenski tip `UnitConverter` opisuje konverzije mernih jedinica. On čuva podatke – faktor konverzije – i ima funkcije članice koji rade s tim podacima.

Konstruktori i instanciranje tipova

Podaci članovi tipa dobijaju vrednosti izradom instance, tj. *instanciranjem*, tipa. Unapred definisani tipovi mogu se instancirati jednostavnim navođenjem literala, kao što su `12` ili `"Zdravo svima"`. Operator `new` pravi instance namenskog tipa. Instancu tipa `UnitConverter` napravili smo i definisali pomoću sledeće naredbe:

```
UnitConverter feetToInchesConverter = new UnitConverter (12);
```

Odmah pošto operator `new` instancira objekat, poziva se *konstruktor* objekta radi inicijalizovanja stanja objekta. Konstruktor se definiše u istom obliku kao metoda, s tom razlikom što su ime metode i njen povratni tip jednaki imenu tipa:

```
public class UnitConverter
{
  ...
  public UnitConverter (int unitRatio) { ratio = unitRatio; }
  ...
}
```

Statički članovi protiv članova instance

Podaci članovi i funkcije članice koje deluju na konkretnu *instancu* tipa, zovu se *članovi instance*. Metoda `Convert` klase (tipa) `UnitConverter` i metoda `ToString` tipa `int` primeri su članova instance. Članovi tipa su ujedno i članovi svake instance tipa.

Podaci članovi i funkcije članice koje ne deluju ni na jednu određenu instancu tipa, nego na sam tip, moraju se označiti 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. Nećete nikad napraviti instancu klase `Console` – istu konzolu dele svi delovi aplikacije.

Da bismo ilustrovali razliku između članova instance i statičkih članova, u narednom kodu polje instance `Name` sadrži vrednost koja pripada jednoj od postojećih instanci klase `Panda` (ta vrednost može biti različita u drugim instancama), dok polje `Population` sadrži vrednost koja je uvek jednaka u svim postojećim instancama klase `Panda`:

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

  public Panda (string n)         // Konstruktor
  {
    Name = n;                     // Dodeljuje vrednost polju instance
    Population = Population + 1;  // Inkrementira statičko polje Population
  }
}
```

Naredni kôd pravi dve instance klase `Panda`, prikazuje njihova imena a zatim prikazuje i ukupan broj pandi:

```
using System;

class Test
{
  static void Main()
```

```
    {
      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
    }
  }
```


Ako pokušate da polju p1.Population ili Panda.Name dodelite vrednost, prouzrokovaćete grešku pri prevođenju.

Rezervisana reč public

Rezervisana reč `public` izlaže (stavlja na raspolaganje) članove jedne klase drugim klasama. U našem primeru, da polje `Name` klase `Panda` nije bilo izričito deklarisano kao `public` (javno), ono bi bilo privatno, a klasa `Test` ne bi mogla da mu pristupa. Deklarisanjem svog člana kao `public`, tip poručuje sledeće: „Ovo hoću da drugi tipovi vide – sve ostalo su moji privatni detalji implementacije“. U objektno orijentisanom modelu kaže se da javni članovi *kapsuliraju* (engl. *encapsulate*) privatne članove tipa.

Konverzije tipova

C# može da pretvara instance jednog tipa u drugi, kompatibilan tip. U postupku konverzije uvek nastaje nova vrednost od postojeće. Konverzije mogu biti *implicitne* ili *eksplicitne*: implicitne konverzije se odvijaju automatski, dok je za eksplicitne konverzije potrebno eksplicitno *pretvaranje* (engl. *casting*). U primeru koji sledi, *implicitno* pretvaramo instancu tipa `int` u tip `long` (koji ima dvostruko veći broj bitova od tipa `int`), ali *eksplicitno* pretvaramo `int` u tip `short` (koji ima upola manji kapacitet od tipa `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-bit ceo broj
```

Implicitne konverzije su dozvoljene kada su ispunjena oba sledeća uslova:

- Kompajler može garantovati da će operacija biti uspešna.
- Pri konverziji se ne gubi ništa od informacije.[1]

Nasuprot tome, *ekplicitne* konverzije su obavezne u jednom od sledećih slučajeva:

- Prevodilac ne može da garantuje uspešnu konverziju.
- Pri konverziji može doći do gubljenja informacija.

(Ako kompajler može da utvrdi da će konverzija *uvek* biti neuspešna, zabranjuje obe vrste konverzija. Konverzije u kojima učestvuju generički tipovi, u određenim uslovima mogu se takođe završiti neuspehom – videti odeljak „Parametri tipa i konverzije“ na strani 115 u poglavlju 3.)

1. Treba imati u vidu da se pri konverziji vrlo velikih vrednosti tipa `long` u `double` gubi deo preciznosti.

Numeričke konverzije koje smo upravo razmotrili, ugrađene su u jezik. C# podržava još i *konverzije referenci* i *konverzije pri pakovanju/raspakivanju* (videti poglavlje 3), kao i *namenske konverzije* (videti odeljak „Preklapanje operatora“ na strani 174 u poglavlju 4). Pošto pri namenskim konverzijama kompajler ne nameće pravila koja smo naveli, moguće je da se loše projektovani tipovi drugačije ponašaju.

Vrednosni tipovi i referentni tipovi

Svaki tip u jeziku C# pripada jednoj od sledećih kategorija:

- Vrednosni tipovi
- Referentni tipovi
- Generički parametri tipa
- Pokazivački tipovi

U ovom odeljku opisaćemo vrednosne tipove i referentne tipove. Generičke parametre tipa razmotrićemo u odeljku „Generičke komponente“ na strani 109 poglavlja 3, a pokazivačkim tipovima bavićemo se u odeljku „Nebezbedan kôd i pokazivači“ na strani 177 poglavlja 4.

Vrednosnim tipovima (engl. *value types*) pripada većina ugrađenih tipova (a to su svi numerički tipovi, zatim tip `char` i tip `bool`), te namenske strukture (tip `struct`) i nabrajanja (tip `enum`).

Referentni tipovi (engl. *reference types*) jesu sve klase i svi nizovi, delegati i interfejsi. (Toj kategoriji pripada i ugrađen tip `string`.)

Suštinsku razliku između vrednosnih i referentnih tipova čini način na koji se oni obrađuju u memoriji.

Vrednosni tipovi

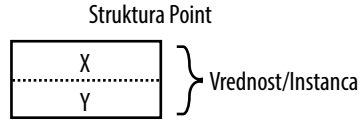
Sadržaj promenljive *vrednosnog tipa* jeste određena konstanta ili vrednost. Na primer, sadržaj ugrađenog vrednosnog tipa `int` čini 32 bita podataka.

Namenski vrednosni tip možete definisati pomoću rezervisane reči `struct` (slika 2-1):

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

ili, u sažetijem obliku:

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



Slika 2-1. Izgled instance vrednosnog tipa u memoriji.

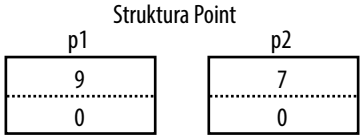
Dodeljivanjem instance vrednosnog tipa drugoj promenljivoj, izvorna instanca se uvek *kopira*.

Na primer:

```
static void Main()
{
  Point p1 = new Point();
  p1.X = 7;
```

```
  Point p2 = p1;             // Dodeljivanje prouzrokuje kopiranje

  Console.WriteLine (p1.X);  // 7
  Console.WriteLine (p2.X);  // 7

  p1.X = 9;                  // Nova vrednost za p1.X

  Console.WriteLine (p1.X);  // 9
  Console.WriteLine (p2.X);  // 7
}
```


Slika 2-2 prikazuje da su promenljive `p1` i `p2` smeštene u memoriju nezavisno jedna od druge.

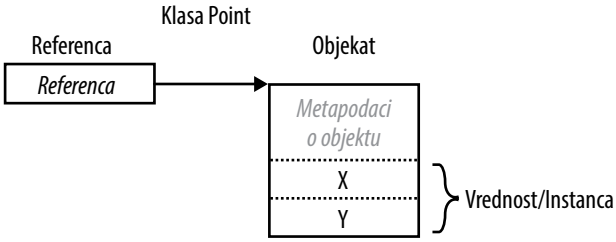


Slika 2-2. Pri operaciji dodeljivanja, instanca vrednosnog tipa se kopira.

Referentni tipovi

Referentni tip je složeniji od vrednosnog i čine ga dva dela: *objekat* i *referenca* na taj objekat. Sadržaj promenljive ili konstante referentnog tipa jeste referenca na objekat koji sadrži vrednost. Evo kako izgleda tip `Point` iz prethodnog primera kada ga napišemo u obliku klase, a ne kao tip `struct` (slika 2-3):

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



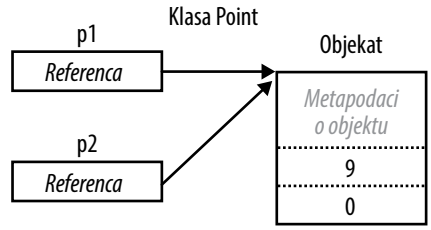
Slika 2-3. Izgled instance referentnog tipa u memoriji.

Pri operaciji dodeljivanja instance referentnog tipa drugoj promenljivoj, kopira se referenca na objekat, a ne sam objekat izvorne instance. To omogućava da više promenljivih upućuje na isti objekat – što obično nije moguće s vrednosnim tipovima. Ako ponovo razmotrimo prethodni primer, s tom razlikom da je sada `Point` klasa, operacija nad promenljivom `p1` sada utiče i na stanje promenljive `p2`:

```
static void Main()
{
  Point p1 = new Point();
  p1.X = 7;

  Point p2 = p1;             // Iz p1 se kopira referenca

  Console.WriteLine (p1.X);  // 7
  Console.WriteLine (p2.X);  // 7
```

```
    p1.X = 9;                  // Nova vrednost za p1.X

    Console.WriteLine (p1.X);  // 9
    Console.WriteLine (p2.X);  // 9
}
```

Slika 2-4 prikazuje da su `p1` i `p2` dve reference koje upućuju na isti objekat.



Slika 2-4. Pri operaciji dodeljivanja, kopira se samo referenca na objekat.

Vrednost null

Referenci se može dodeliti literal `null`, koji znači da takva referenca ne upućuje ni na jedan objekat:

```
class Point {...}
...

Point p = null;
Console.WriteLine (p == null);   // True

// Sledeći red prouzrokuje grešku pri izvršavanju
// (izuzetak NullReferenceException):

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

Nasuprot tome, vrednosni tip ne može nikad imati vrednost `null`:

```
struct Point {...}
...

Point p = null;  // Greška pri prevođenju
int x = null;    // Greška pri prevođenju
```

C# ima konstrukt nazvan *tip koji prihvata null*, (engl. *nullable*) čija je namena predstavljanje vrednosti `null` u slučaju vrednosnih tipova (videti odeljak „Tipovi koji prihvataju `null`“, na strani 152 poglavlja 4).

Potrošnja memorije

Instance vrednosnih tipova zauzimaju tačno onoliko memorije koliko je potrebno za smeštanje vrednosti iz njihovih polja. U našem primeru, `Point` zauzima osam bajtova memorije:

```
struct Point
{
  int x;  // 4 bajta
  int y;  // 4 bajta
}
```