

*Struktura* (engl. *struct*) je jednostavan korisnički definiran tip, lakša alternativa klasi. Strukture su slične klasama u toliko što mogu sadržati konstruktore, svojstva, metode, polja, operatore, ugniježdene tipove i indekse (pogledajte poglavlje 9).

Postoje i značajne razlike između klasa i struktura. Na primjer, strukture ne podržavaju nasljeđivanje niti destruktore. Što je još važnije, iako je klasa referentni tip, struktura je vrijednosni tip (više informacija o klasama i tipovima potražite u poglavlju 3). Strukture su stoga korisne za predstavljanje objekata koji ne zahtijevaju semantiku referenci.

Općeprihvaćeno stajalište je da se strukture trebaju koristiti samo za tipove koji su mali, jednostavni i po ponašanju i karakteristikama slični ugrađenim tipovima.



*Napomena za C++ programere:* značenje konstrukcije strukture u C# uvelike je drugačije od onog u C++. Struktura je u C++ potpuno jednaka klasi, osim što je podrazumijevana postavka vidljivosti (javna nasuprot privatnoj) drugačija. U C# strukture su vrijednosni tipovi, dok su klase referentni tipovi, a strukture u C# imaju i druga ograničenja koja su opisana u ovom poglavlju.

Strukture su nešto učinkovitije po upotrebi memorije u poljima (za više informacija pogledajte poglavlje 9). One, međutim, mogu biti manje učinkovite kada se koriste u nekim kolekcijama. Kolekcije koje uzimaju objekte očekuju reference, a strukture moraju biti zapakirane. Pakiranje i raspakiravanje usporava izvedbu pa klase mogu biti učinkovitije u većim kolekcijama.

U ovom ćete poglavlju naučiti kako se strukture definiraju i kako se s njima radi te kako se za inicijalizaciju njihovih vrijednosti mogu koristiti konstruktori.

## Definiranje struktura

Sintaksa za deklariranje strukture gotovo je identična onoj za definiranje klase:

```
[atributi] [modifikator pristupa] struct identifikator [:popis sučelja]  
{ članovi }
```

U primjeru 7-1 prikazana je definicija strukture. Location predstavlja točku na dvodimenzionalnoj površini. Primijetit ćete kako se struktura Location deklarira isto kao što bi se deklarirala klasa, osim što se koristi ključna riječ struct. Također možete primijetiti kako konstruktor Location uzima dvije cjelobrojne vrijednosti i dodjeljuje ih članovima instance xVal i yVal. Koordinate x i y za Location su deklarirane kao svojstva.

#### *Primjer 7-1. Stvaranje strukture*

```
#region Using directives

using System;
using System.Collections.Generic;
using System.Text;

#endregion

namespace CreatingAStruct
{
    public struct Location
    {
        private int xVal;
        private int yVal;

        public Location( int xCoordinate, int yCoordinate )
        {
            xVal = xCoordinate;
            yVal = yCoordinate;
        }

        public int x
        {
            get
            {
                return xVal;
            }
            set
            {
                xVal = value;
            }
        }

        public int y
        {
            get
            {
                return yVal;
            }
            set
            {
                yVal = value;
            }
        }
    }
}
```

### Primjer 7-1. Stvaranje strukture (nastavak)

```
        public override string ToString()
        {
            return ( String.Format( "{0}, {1}", xVal, yVal ) );
        }
    }

    public class Tester
    {
        public void myFunc( Location loc )
        {
            loc.x = 50;
            loc.y = 100;
            Console.WriteLine( "In MyFunc loc: {0}", loc );
        }
        static void Main()
        {
            Location loc1 = new Location( 200, 300 );
            Console.WriteLine( "Loc1 location: {0}", loc1 );
            Tester t = new Tester();
            t.myFunc( loc1 );
            Console.WriteLine( "Loc1 location: {0}", loc1 );
        }
    }
}
```

Za razliku od klasa, strukture ne podržavaju nasljeđivanje. One su implicitno izvedene iz object (isto kao i svi tipovi u C#, uključujući ugrađene tipove), ali ne mogu nasljeđivati od drugih klasa niti struktura. Strukture su također implicitno *zapečaćene* (to jest, nijedna klasa niti struktura se ne može izvesti iz strukture). Strukture, međutim, poput klasa mogu implementirati više sučelja. Ostale razlike uključuju:

#### *Nema destruktora niti prilagođenog podrazumijevanog konstruktora*

Strukture ne mogu sadržati destruktore niti prilagođene konstruktore bez parametara (podrazumijevane). Ako konstruktor ne postoji, CLR će inicijalizirati strukturu i sva polja postaviti na nulu. Ako pak navedete konstruktor koji nije zadan, CLR inicijalizacija se neće pokrenuti te morate eksplicitno inicijalizirati sva polja.

#### *Nema inicijalizacije*

U strukturi ne možete inicijalizirati polje instance. Stoga, nije dopušteno napisati:

```
private int xVal = 50;
private int yVal = 100;
```

iako bi to bilo u redu da se radi o klasi.

Strukture su projektirane da budu jednostavne i lake za korištenje. Dok privatni podaci člana promoviraju sakrivanje i učahurivanje podataka, neki programeri drže kako je to prevelik napor za strukture. Oni podatke člana označavaju kao javne i tako

pojednostavniju implementaciju strukture. Drugi programeri misle kako svojstva omogućavaju jasno i jednostavno sučelje te da dobra programerska praksa nalaže sakrivanje podataka čak i u jednostavnim objektima. Zahvaljujući novoj mogućnosti refaktoriranja u Visual Studiju javne varijable možete jednostavno pretvoriti u privatne sa pridruženim javnim svojstvima. Varijablu jednostavno pritisnete desnom tipkom miša i odaberete Refactor → Encapsulate Field. Visual Studio će javnu varijablu pretvoriti u privatnu i stvoriti svojstvo s pristupnicima get i set.

## Stvaranje struktura

Instancu strukture možete stvoriti koristeći ključnu riječ `new` u iskazu dodjeljivanja, isto kao da se radi o klasi. U primjeru 7-1 klasa `Tester` stvara instancu `Location` na sljedeći način:

```
Location loc1 = new Location(200,300);
```

Ovdje je nova instanca nazvana `loc1` i prosljeđuju joj se dvije vrijednosti, 200 i 300.

## Strukture kao vrijednosni tipovi

Definicija klase `Tester` u primjeru 7-1 sadrži strukturu objekta\* `Location` (`loc1`) koja je stvorena s vrijednostima 200 i 300. Sljedeći red koda poziva konstruktora `Location`:

```
Location loc1 = new Location(200,300);
```

Zatim se poziva `WriteLine()`:

```
Console.WriteLine("Loc1 location: {0}", loc1);
```

`WriteLine()` očekuje objekt, ali, naravno, `Location` je struktura (vrijednosni tip). Prevoditelj automatski pakira strukturu (kao što bi to učinio s bilo kojim drugim vrijednosnim tipom) i taj se zapakirani objekt prosljeđuje do `WriteLine()`. Za zapakirani objekt poziva se `ToString()` a kako struktura (implicitno) nasljeđuje od `object`, ona može odgovoriti polimorfno, premošćujući metodu na isti način na koji bi to učinio bilo koji drugi objekt:

```
Loc1 location: 200, 300
```



Pakiranje možete izbjeći tako da prethodni odjeljak promijenite u:

```
Console.WriteLine("Loc1 location: {0}",  
loc1.ToString());
```

Pakiranje se izbjegava izravnim pozivanjem metode `ToString` za varijablu vrijednosnog tipa gdje vrijednosni tip pruža premošćavanje metode `ToString`.

\* U ovoj knjizi termin objekt koristim i za referentne tipove i za vrijednosne tipove. U svijetu objektno orijentiranog programiranja postoji neslaganje oko takve prakse, ali ja se tješim činjenicom da je Microsoft vrijednosne tipove implementirao kao da su naslijedili od korijenske klase `Object` (i stoga za bilo koji vrijednosni tip, uključujući ugrađene tipove poput `int`, možete pozivati sve metode `Object`).

Strukture su, međutim, vrijednosni objekti i kad se proslijeđuju do funkcije oni se proslijeđuju po vrijednosti - kao što možete vidjeti u sljedećem redu koda u kojem se objekt `loc1` proslijeđuje metodi `myFunc()`:

```
t.myFunc(loc1);
```

U metodi `myFunc()` nove se vrijednosti dodjeljuju `x` i `y` i te se nove vrijednosti ispisuju:

```
Loc1 location: 50, 100
```

Kad se vratite na pozivajuću funkciju (`Main()`) i ponovno pozovete `WriteLine()` vrijednosti ostaju nepromijenjene:

```
Loc1 location: 200, 300
```

Struktura je proslijeđena kao vrijednosni objekt te je u `myFunc()` stvorena kopija. Pokušajte deklaraciju promijeniti u `class`:

```
public class Location
```

i ponovno pokrenite provjeru. Generira se sljedeći izlaz:

```
Loc1 location: 200, 300  
In MyFunc loc: 50, 100  
Loc1 location: 50, 100
```

Ovaj put objekt `Location` ima semantiku reference. Stoga, kad se vrijednosti promijene u `myFunc()`, one se promijene u stvarnom objektu u `Main()`.\*

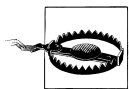
## Stvaranje struktura bez ključne riječi `new`

Budući da je `loc1` struktura (a ne klasa), stvorena je na stogu. Stoga, kad je u primjeru 7-1 pozvan operator `new`:

```
Location loc1 = new Location(200,300);
```

rezultirajući objekt `Location` je stvoren na stogu.

Operator `new` poziva konstruktor `Location`. Međutim, za razliku od klase, struktura se može stvoriti bez korištenja operatora `new`. To je u sklad s načinom definiranja varijabli ugrađenih tipova (poput `int`), kao što je prikazano u primjeru 7-2.



Opomena: u ovom primjeru je prikazan način stvaranja strukture bez korištenja operatora `new` jer se `C#` i `C++` po tome razlikuju. Stvaranje struktura bez ključne riječi `new` donosi malo prednosti i na taj se način mogu stvoriti programi koji su manje razumljivi, više podložni pogreškama i teži za održavanje. Nastavite na vlastitu odgovornost.

\* Drugi način za rješavanje ovog problema je upotreba ključne riječi `ref` (kao što je objašnjeno u poglavlju 4) koja dopušta da vrijednosni tip proslijedite po referenci.

*Primjer 7-2. Stvaranje strukture bez ključne riječi new*

```
#region Using directives

using System;
using System.Collections.Generic;
using System.Text;

#endregion

namespace StructWithoutNew
{
    public struct Location
    {
        public int xVal;
        public int yVal;

        public Location( int xCoordinate, int yCoordinate )
        {
            xVal = xCoordinate;
            yVal = yCoordinate;
        }
        public int x
        {
            get
            {
                return xVal;
            }
            set
            {
                xVal = value;
            }
        }

        public int y
        {
            get
            {
                return yVal;
            }
            set
            {
                yVal = value;
            }
        }

        public override string ToString()
        {
            return ( String.Format( "{0}, {1}", xVal, yVal ) );
        }
    }

    public class Tester
    {
        static void Main()
    }
}
```

Primjer 7-2. Stvaranje strukture bez ključne riječi new (nastavak)

```
{
    Location loc1;          // Nema poziva konstruktora

    loc1.xVal = 75;        // Inicijalizira članove
    loc1.yVal = 225;
    Console.WriteLine( loc1 );
}
}
```

U primjeru 7-2 lokalne varijable se inicijaliziraju izravno prije pozivanja metode loc1 i prije proslijeđivanja objekta do WriteLine():

```
loc1.xVal = 75;
loc1.yVal = 225;
```

Kad bi jednu od dodjela smjestili u komentar i ponovno preveli kod:

```
static void Main()
{
    Location loc1;
    loc1.xVal = 75;
    // loc1.yVal = 225;
    Console.WriteLine(loc1);
}
```

došlo bi do pogreške prevoditelja:

```
Use of unassigned local variable 'loc1'
```

Kad dodijelite sve vrijednosti, možete im pristupiti preko svojstava x i y:

```
static void Main()
{
    Location loc1;
    loc1.xVal = 75;          // Dodjeljuje varijablu članicu
    loc1.yVal = 225;        // Dodjeljuje varijablu članicu
    loc1.x = 300;           // Koristi svojstvo
    loc1.y = 400;           // Koristi svojstvo
    Console.WriteLine(loc1);
}
```

Budite oprezni prilikom korištenja svojstava. Iako ona daju podršku za učajurivanje tako što stvarne vrijednosti čine privatnima, sama su svojstva zapravo metode članice, a metodu članicu ne možete pozvati dok ne inicijalizirate sve varijable članice.