




Modulul 7

Folosirea variabilelor tip referință

În acest modul, veți învăța cum se folosesc tipurile referință în C#. Veți învăța despre un număr de variabile referință, cum ar fi **string**, care sunt incluse în limbajul C#.

Veți învăța, de asemenea, despre ierarhia de clase **System.Object** și despre tipul **Object** în particular, pentru a înțelege mai bine cum diferitele tipuri referință sunt legate între ele și cu tipurile valoare. Veți învăța să converțiți date între tipuri referință, folosind conversii implicite și explicite. Se va discuta, de asemenea, despre conversiile de tip **boxing** și **unboxing**, folosite pentru a converti date între tipuri referință și tipuri valoare.




itacad
you@technology

Microsoft®
.NET

Overview

- ▶ Tipuri referință
- ▶ Variabile de tip referință
- ▶ Tipul *Object*
- ▶ Boxing și Unboxing



2

- La finalul acestui modul, veți putea să:
- Descrieți diferențele importante între tipurile referință și tipurile valoare
 - Faceți conversii implicite și explicite între tipuri referință
 - Realizați conversii de tip **boxing** și **unboxing** între tipuri referință și valoare.



itacad
you@technology

Microsoft
.NET

Folosirea variabilelor de tip referință

- ▶ Comparație între tipuri valoare și tipuri referință
- ▶ Declararea și eliberarea variabilelor referință
- ▶ Referințe ce nu sunt valide
- ▶ Operatorii de comparație aplicați pe tipuri valoare și pe tipuri referință
- ▶ Referințe multiple către același obiect
- ▶ Folosirea referințelor ca parametrii pentru metode

3

Tipurile referință sunt foarte importante în limbajul C#. Ele vă permit să scrieți aplicații complexe și puternice și să folosiți platforma la capacitățile ei maxime.

Comparație între tipuri valoare și tipuri referință

Tipuri valoare

- Variabila conține direct valoarea
- Exemple:
char, int

```
int valType;  
valType = 42;
```

42

Tipuri referință

- Variabila conține doar o referință către locul unde sunt reținute datele
- Datele sunt reținute într-o zonă separată de memorie

```
string refType;  
refType = "Hello"
```

•

→ Hello

C# suporta tipuri de bază, cum ar fi **int**, **long** și **bool**. Aceste tipuri mai sunt numite și tipuri valoare. C# suporta, de asemenea, și tipuri mai complexe și mai puternice, numite tipuri referință.

Tipuri valoare

Variabilele tip valoare sunt tipurile de bază suportate de limbaj, precum **char** sau **int**. Ele reprezintă cele mai simple tipuri din C#. Variabilele de acest tip conțin direct datele în variabilă.

Tipuri referință

Variabilele de tip referință conțin doar o referință cu privire la zona unde sunt stocate datele și nu datele direct. Ați avut deja de-a face cu astfel de tipuri de variabile. Vectorii, stringurile și excepțiile sunt toate tipuri referință, ce sunt implementate în compilatorul C# și în platforma .NET. Clasele, fie deja definite, fie definite de utilizator, sunt, de asemenea, variabile tip referință.

itacad
you@technology

Declararea și eliberarea variabilelor referință

- ▶ Declararea și inițializarea variabilelor referință

```
coordinate c1;
c1 = new coordinate();
c1.x = 6.12;
c1.y = 4.2;
```

•

→

6.12 | 4.2

- ▶ Eliberarea variabilelor referință

```
c1 = null;
```

•

✗

6.12 | 4.2

5

Pentru a folosi variabile de tip referință, trebuie întâi văzut cum se declară și cum se eliberează o astfel de variabilă.

Declararea variabilelor de tip referință

Declararea unei astfel de variabile se face asemănător cu variabilele de tip valoare.

```
coordinate c1;
```

Exemplul precedent declară o variabilă **c1** ce poate păstra o referință către un obiect de tip **coordinate**. Totuși, variabila nu este inițializată.

Pentru a inițializa un obiect de tip **coordinate**, trebuie folosit operatorul **new**. Acesta va crea un nou obiect și va întoarce referința sa, ce va fi salvată în variabila referință.

```
coordinate c1 = new coordinate();
```

După crearea unui obiect în memorie a cărui referință este ținută în **c1**, puteți accesa diverși membri ai obiectului folosind operatorul **punct (.)** ca în exemplu:

```
c1.x = 6.12
c1.y = 4.2
```

Exemplu de declarare a unei variabile referință

Clasele sunt tipuri referință. Următorul exemplu arată cum se declară o clasă definită de utilizator numită **coordinate**. Pentru simplitate, clasa are doar 2 membrii publici, variabilele **x** și **y**:

```
class coordinate
{
    public double x = 0.0;
    public double y = 0.0;
}
```

Eliberarea variabilelor referință

După ce ați asignat o referință către un nou obiect, variabila referință va continua să referențieze acel obiect până când i se asignează o referință către un nou obiect.

C# definește o valoare specială numită **null**. O variabilă referință conține **null** când nu pointează către nici un obiect. Pentru a elibera o variabilă referință, puteți să îi asigurați explicit valoarea **null**.


itacad
you@technology

Microsoft®
.NET

Referințe ce nu sunt valide

- ▶ În cazul unor referințe ce nu sunt valide
 - ▶ Nu veți putea accesa membrii
- ▶ Referințe invalide detectate la compilare
 - ▶ Compilatorul va detecta utilizarea de referințe neinițializate
- ▶ Referințe invalide apărute la *run-time*
 - ▶ Se va arunca o excepție în momentul în care va fi utilizată o referință care a fost ștearsă

7

Puteți accesa membrii unei obiect printr-o variabilă referință doar dacă variabila referință a fost setată să poarte către o zonă de memorie validă. Dacă referința nu este validă, nu puteți să accesați membrii obiectului.

În unele cazuri, compilatorul va observa aceste probleme și le va raporta la compilare. În alte cazuri, problemele vor apărea la *run-time* și vor trebui să fie tratate atunci.

Referințe invalide la compilare

Una dintre situațiile pe care le va observa compilatorul este folosirea unei variabile înainte inițializării ei. În acest caz, veți primi un mesaj de eroare de forma: "Use of unassigned local variable c1" dacă încercați să compilați următorul cod:


```
coordinate c1;  
c1.x = 6.12           // va genera eroare la compilare
```

Referințe invalide la run-time

În general, nu este posibil să se observe când o variabilă referință nu este validă. De aceea, C# va verifica valoarea unei variabile referință înainte să fie accesată, pentru a garanta că nu este **null**.


Dacă încercați să folosiți o variabilă referință a cărei valoare este **null**, se va arunca o excepție de tipul **NullReferenceException**. Se poate verifica folosind try catch dacă a apărut o astfel de problemă.

```
try
{
    c1.x = 45;
}
catch (NullReferenceException)
{
    Console.WriteLine("c1 has a null value");
}
```



itacad
 you@technology

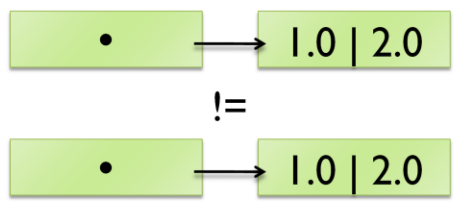
 Microsoft
.NET

Operatorii de comparație



- ▶ Aplicați asupra tipurilor valoare
 - ▶ Vor compara valorile

- ▶ Aplicați asupra tipurilor referință
 - ▶ Vor compara referințele, nu valorile
 - ▶ Se verifică dacă referințele au asociate sau nu aceeași zonă de memorie



9

Operatorii de egalitate (==) și inegalitate (!=) s-ar putea să nu funcționeze cum v-ați aștepta pentru variabile de tip referință.

Compararea tipurilor valoare

Pentru aceste tipuri, compararea valorilor se face fără probleme folosind acești operatori

Compararea tipurilor referință

Pentru aceste tipuri, operatorii vor verifica dacă cele două referințe pointează către același obiect. **NU** se va compara conținutul.

Obs: pentru tipul **string** (tip referință), operatorii vor compara valoarea, nu referința.

Un exemplu pentru a exemplifica: două variabile de tip **coordinate** sunt create și inițializate cu aceeași valoare:

```
coordinate c1= new coordinate( );
coordinate c2= new coordinate( );
c1.x = 1.0;
c1.y = 2.0;
c2.x = 1.0;
c2.y = 2.0;
```

```
if (c1 == c2)
{
    Console.WriteLine("Same");
}
else
{
    Console.WriteLine("Different");
}
```

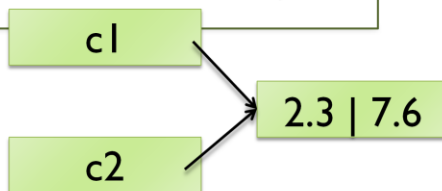
La rulare, se va afișa mesajul “Different” deși obiectele pe care le referențiază c1 și c2 au aceeași valoare.

Obs: Pentru tipuri referință, nu se pot folosi ceilalți operatori operaționali (<, >, <=, >=).

Referințe multiple către același obiect

- ▶ Două referințe pot pointa către aceeași valoare
 - ▶ Două căi de a accesa aceeași valoare

```
coordinate c1 = new coordinate( );
coordinate c2;
c1.x = 2.3; c1.y = 7.6;
c2 = c1;
Console.WriteLine(c1.x + " , " + c1.y);
Console.WriteLine(c2.x + " , " + c2.y);
```



11

Două variabile referință pot pointa către același obiect. Asta înseamnă că puteți scrie date folosind o referință și citi aceste date folosind cealaltă referință.

Referințe multiple către același obiect


În exemplul din slide, variabila c1 este inițializată pentru a pointa către o nouă instanță de clasă și membrii săi sunt inițializați. Apoi c1 este copiată în c2. La final, se afișază conținutul membrilor variabilelor.

Rezultatul va fi acesta:

2.3 , 7.6

2.3 , 7.6

Asignarea lui c2 la valoarea lui c1 copiază valoarea referinței astfel încât ambele variabile referențiază aceeași instanță. Astfel, valorile afișate vor fi aceleași.

 itacad
you@technology

Microsoft
2.NET

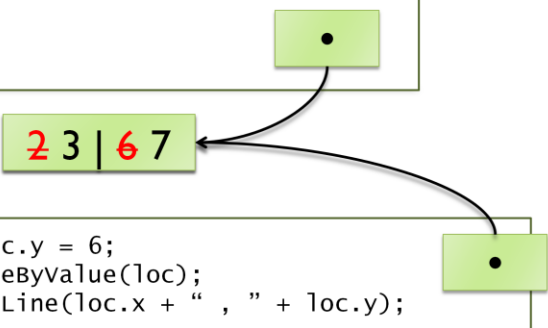
Folosirea referințelor ca parametri pentru metode

- ▶ Referințele pot fi trimise ca parametri metodelor
 - ▶ Datele ce sunt referențiate pot să fie modificate în metode

```
static void PassCoordinateByValue (coordinate c)
{
    c.X++; c.Y++;
}
```

2 3 | 6 7

```
loc.x = 2; loc.y = 6;
PassCoordinateByValue(loc);
Console.WriteLine(loc.x + " , " + loc.y);
```



12

Folosirea referințelor ca parametri pentru metode

Se pot pasa variabile referință în metode ca parametri folosind oricare dintre mecanismele disponibile:

- prin valoare
- prin referință
- ca parametru de output

Următorul exemplu arată o metodă ce primește trei referințe ale unor obiecte **coordinate**. Prima este trimisă prin valoare, a doua prin referință, iar a treia ca parametru de output. Valoarea întoarsă de metodă este o referință la un obiect **coordinate**.

```
static coordinate Example( coordinate ca, ref coordinate cb, out coordinate cc)
{
    // ...
}
```

Trimiterea referințelor prin valoare

Când folosiți această metodă de trimitere a parametrilor, metoda va primi o copie a referinței. Asta înseamnă că, pe durata apelului, vor exista două referințe către același obiect (una în metoda ce a trimis respectivul parametru și a doua în metoda ce l-a primit). Astfel, ca în slide, modificările făcute asupra obiectului se vor păstra la finalul metodei.

Modificările aduse referinței, însă, nu se vor păstra:

```
static void PassCoordinateByValue(coordinate c)
{
    c = new coordinate( );
    c.x = c.y = 22.22;
}
static void Main()
{
    coordinate loc = new coordinate( );
    PassCoordinateByValue(loc);
    Console.WriteLine(loc.x + " , " + loc.y);    // 0.0 , 0.0 (valorile
default)
}
```

Trimiterea referințelor prin referință

Când trimiteți o variabilă referință ca parametru **ref**, metoda va primi efectiv variabila referință. Spre deosebire de cazul anterior, modificările aduse variabilei se vor păstra după terminarea apelului metodei curente. Metoda nu va mai face propria copie a variabilei, ci va lucra direct cu cea din metoda care a făcut apelul:

```
static void PassCoordinateByRef(ref coordinate c)
{
    c = new coordinate( );
    c.x = c.y = 33.33;
}
static void Main()
{
    coordinate loc = new coordinate( );
    PassCoordinateByRef(ref loc);
    Console.WriteLine(loc.x + " , " + loc.y);    // 33.33 , 33.33
}
```

Trimiterea referințelor ca parametri de output

Comportamentul va fi similar cu cel de la trimiterea prin referință. Nu se va face o copie locală, ci se va lucra direct pe variabila referință. Diferența față de trimiterea prin referință este aceea că, în acest caz, variabila trebuie inițializată înainte de a putea fi folosită.

```
static void PassCoordinateByOut(out coordinate c)
{
    c = new coordinate( );
    c.x = c.y = 44.44;
}
static void Main()
{
    coordinate loc;
    PassCoordinateByOut(out loc);
    Console.WriteLine(loc.x + " , " + loc.y);    // 44.44, 44.44
}
```

Observație: Variabilele tip referință nu rețin direct valoarea datelor. Acest lucru se aplică și când sunt trimise ca parametri. Chiar dacă am trimis respectiva variabilă referință ca parametru valoare, modificările aduse obiectului pe care îl referențiază se vor păstra.

Tipul **string** este referință

- ▶ Valorile de tip *string* sunt referințe către locul unde se găsește șirul de caractere
 - ▶ Fie că este obținut de la tastatura
 - ▶ Fie că este scris între ghilimele în cod
- ▶ Pentru toate șirurile de caractere din cod se crează câte o referință la *run-time*
- ▶ Valoarea indicată de o referință *string* nu poate fi modificată
 - ▶ Pentru fiecare șir de caractere nou se crează o altă referință

14

O variabilă *string* neinițializată va genera aceeași eroare ca orice altă variabilă de tip referință în momentul în care se dorește utilizarea valorii referențiate.

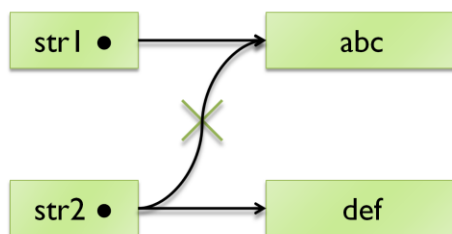
```
string str;
Console.WriteLine(str); //va genera eroare la compilare
```

Deoarece *string* este un tip referință, asignarea valorilor se face la fel de repede indiferent de mărimea șirului de caractere reprezentat. Acest lucru se datorează faptului că se copiază referința și nu conținutul.

Pentru a evita problemele care apar în momentul în care 2 variabile au setate aceeași referință și una dintre ele modifică valoarea referențiată, tipul *string* este imutabil. Acest lucru înseamnă că, de fiecare dată când se dorește alterarea șirului de caractere, în realitate se creează unul nou, împreună cu referința asociată.

Funcționarea tipului **string**

```
string str1 = "abc";  
string str2 = str1;  
str2 = "def";
```



15

Șirul "abc" este înlocuit cu o referință corespunzătoare. Referința este atribuită variabilei **str1**. Variabila **str2** copiază această referință. Șirul "def" este înlocuit cu o referință corespunzătoare. Referința este atribuită variabilei **str2**. În acest moment, cele 2 variabile indică zone de memorie diferite.

Codul poate fi gândit în următorul fel:

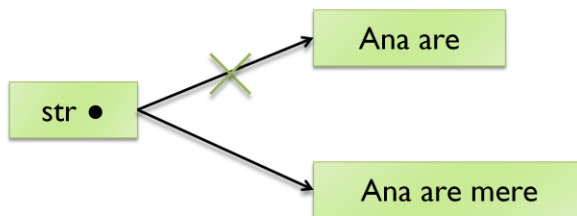
```
string str1 = new string("abc");  
string str2 = str1;  
str2 = new string("def");
```

Pentru a îmbunătăți acest comportament, șirul de caractere indicat de referință poate fi doar citit, nu și modificat.

```
string str = "Ana are mere.";  
str[5] = 'r'; //va genera o eroare de compilare
```

Funcționarea tipului **string** (2)

```
string str = "Ana are";  
string str = str + " mere";  
Console.WriteLine(str); //se va afisa: Ana are mere
```



16

La început, variabila **str** va pointa către șirul de caractere "Ana are". Operatorul + concatenează cele 2 șiruri de caractere, cel delimitat de **str** și " mere", și pune rezultatul într-o nouă zonă de memorie. Rezultatul concatenării este o nouă referință către șirul de caractere "Ana are mere".

La finalul execuției codului de mai sus, în cadrul programului, vor exista 2 șiruri de caractere în zone de memorie diferite, prin urmare 2 referințe diferite. Prima dintre ele nu mai este utilizată. Într-un scenariu asemănător, prima referință putea să fie stocată în continuare în altă variabilă.

Tipul Object

- ▶ Poate stoca referințe indiferent de tipul acestora
- ▶ Există situații în care trebuie făcute prelucrări specifice referințelor pe variabile de tip valoare
 - ▶ Atribuire cu valori complexe reduc performanța aplicației
 - ▶ Se poate crea o referință către acea valoare complexă
- ▶ Conversiile de date fac posibilă asocierea de referințe pentru valori standard
 - ▶ Se pot crea referințe către: **int, float, struct**

17

Object reprezintă tipul cel mai simplu din cadrul platformei .NET. Orice fel de referință poate fi salvată prin intermediul unei variabile de tip *object*.

Variabilele de tip referință au câteva avantaje care pot fi utile inclusiv pentru variabilele de tip valoare. De exemplu, interschimbarea valorilor a 2 elemente se va face mai rapid prin intermediul unei referințelor, deoarece valorile vor fi pastrate neschimbate. În schimb, accesări succinte ale valorii stocate de o referință poate încetini viteza de execuție a programului. În astfel de situații, se va folosi o variabilă de tip valoare pentru a stoca, în mod direct, datele.

Este posibil a se genera o referință către tipuri elementare cum ar fi: **int, float, struct**. Astfel, se pot combina avantajele referințelor împreună cu avantajele valorilor standard.

Referințe sub formă de obiect

- Poate fi considerată o referință generală
 - Toate valorile pot fi păstrate sub formă de referințe *obiect*

```
coordinate c1 = new coordinate( );
c1.x = 2; c1.y = 6;

object obj = c1;
```

- Atunci când este nevoie de valoarea asociată, referința trebuie convertită la tipul inițial

```
coordinate c2 = (coordinate)obj;
c2.x = 5;
```



18

În exemplul de mai sus, variabila **obj** va salva aceeași referință ca variabila **c1**, diferența fiind faptul că se vor pierde informații referitoare la posibilitatea de accesare a zonei de memorie referențiate.

Pentru recăpătarea funcționalităților, referința *object* trebuie convertită înapoi la tipul original. În acest sens, variabila **c2** va putea fi folosită în mod identic cu variabila **c1**.

Modificarea adusă de ultima linie de cod, va fi vizibilă de către toate cele 3 variabile de tip referință. În final, toate vor indica valoarea cu x=5 și y=6.

Operatorii **is** și **as**

- ▶ Sunt folosiți pentru a identifica dacă o referință este convertibilă la o referință de alt tip
- ▶ Sunt utili atunci când există referințe de tip *object*

```
Bird a = new Bird();
object obj = a;
```

- ▶ Operatorul **is** verifică dacă referința poate fi convertită corespunzător
- ▶ Operatorul **as** încearcă să realizeze conversia fără a arunca excepții în caz de eșec

20

Acești operatori sunt des folosiți pentru conversii între referințe. Ei stabilesc dacă o referință de un anumit tip poate fi convertit la altul.

Există situații în care o referință poate fi convertită la mai multe tipuri diferite, nu neapărat la *object*. Acest lucru este posibil în cazul în care structurile acelor referințe au o bază comună. Astfel de situații se întâlnesc în urma procedurii de moștenire a claselor.

Valoarea de tipul *Bird* este stocată în variabila **obj**. În acest moment, nu se poate spune nimic despre structura de date existentă în locul indicat de către referință. Pentru identificarea acestuia se vor folosi operatorii **is** și **as**.

Operatorul **is** este folosit pentru a testa dacă referința conține date care pot fi interpretate specific unei referințe de alt tip, pe când operatorul **as** încearcă să interpreteze zona de memorie folosind structurile de date asociate tipului specificat.

Operatorul **is**

- Va întoarce **true** dacă o conversie poate fi făcută

```
Bird b = new Bird();
Object obj = b;

if (obj is Bird)
{
    b = (Bird) obj; //se poate face cast
}
else
{
    Console.WriteLine("Not a bird");
}
```

21

O variantă pentru a vedea dacă o conversie poate fi făcută este încercarea conversiei și apoi prinderea excepției **InvalidCastException**, dacă apare. Există însă și alte variante pentru a rezolva această problemă, cum ar fi operatorul **is**.

Acest operator poate fi folosit pentru a testa tipul obiectului, fără a face conversia. Operatorul **is** va întoarce **true** dacă valoarea din stânga nu este **null** și dacă, la încercarea unui cast către tipul din dreapta, această operație s-ar executa fără a arunca excepții. **is** va întoarce **false** altfel.


itacad
you@technology

Microsoft
.NET

Operatorul **as**

- ▶ Face conversia între tipuri, ca un cast
- ▶ Dacă apare eroare
 - ▶ Întoarce **null**
 - ▶ Nu va arunca excepție

```
Bird b = obj as Bird; //conversie

if (b == null)
{
    Console.WriteLine("Not a bird");
}
```

22

Se poate folosi operatorul **as** pentru a face conversia între tipuri

Exemplu

Următoare instrucțiune execută conversia referinței *obj* la o valoare ce referențiază o clasă de tip **Bird**, la run-time se verifică automat dacă conversia este acceptabilă.

```
b = obj as Bird;
```

Tratarea erorilor

Operatorul **as** diferă de operatorul de cast prin felul în care sunt tratate erorile. Dacă, în exemplul anterior, variabila *obj* nu putea să fie convertită la o referință de tipul clasei **Bird**, lui *b* i se atribuia valoarea **null** și programul continua. Acest operator nu aruncă niciodată excepții.

Deși nu se aruncă excepții, dacă încercăm să accesăm variabila *b* se va arunca o excepție de tipul **NullReferenceException**. De aceea, este recomandat să verificați valoarea întoarsă de operatorul **as**.

Conversiile și tipul **object**

- ▶ Orice referință poate fi convertită cu succes către *object*
 - ▶ Operatorul **is** aplicat tipului *object* va întoarce mereu **true**
- ▶ Orice referință poate fi asignată unei variabile *object*
- ▶ Valoarea unei variabile de tipul *object* poate fi asignată oricărei variabile tip referință
 - ▶ Conversia la tipul variabilei trebuie să se execute cu succes

```
object ox;
ox = a;
ox = (object) a;
ox = a as object;
```

```
b = (Bird) ox;
b = ox as Bird;
```

23

Toate tipurile referință au la bază tipul **object**. Asta înseamnă că orice referință poate fi stocată într-o variabilă de tip *object*.

Tipul **object** este baza pentru toate celelalte clasele

Tipul *object* este baza pentru toate tipurile referință

Orice referință poate fi asignată unei variabile de tip **object**

Datorită faptului că toate clasele sunt direct sau indirect bazate pe tipul *object*, puteți atribui orice referință unei variabile ce are tipul *object*, folosind cast implicit sau explicit.

```
object ox;
ox = a;
ox = (object) a;
ox = a as object;
```

Valoarea unei variabile de tipul **object** poate fi asignată oricărei variabile tip referință

Pentru a face acest lucru, cast-ul trebuie făcut cu succes. Țineți minte că, la run-time, se va verifica, la fiecare operație de atribuire, dacă o variabilă de tipul corect a fost furnizată.

```
b = (Bird) ox;
b = ox as Bird;
```

Exemplul anterior poate fi rescris, pentru a verifica eventualele erori ce pot să apară

```
try
{
    b = (Bird) ox;
}
catch (InvalidCastException)
{
    Console.WriteLine("Cannot convert to Bird");
}
b = ox as Bird;
if (b == null)
    Console.WriteLine("Cannot convert to Bird");
```


itacad
you@technology

Microsoft
.NET

Boxing și Unboxing

- ▶ Sistem de tipuri unificat
- ▶ Boxing (conversie valoare către *object*)
- ▶ Unboxing (conversie *object* către valoare)
- ▶ Apelare metode ale clasei *Object* pentru tipuri valoare

```
int p = 42;
object box;
box = p;
```

42

↕

•

→

42

```
p = (int) box;
```

25

C# poate converti tipuri valoare în referințe către obiecte și referințe către obiecte în tipuri valoare.

Unified Type System

C# are un sistem de tipuri unificat, ce permite tipurilor valoare să fie convertite în tipuri obiect și referințelor către obiecte să fie convertite în tipuri valoare.

Tipurile valoare, cum ar fi **int** sau **bool**, pot astfel să fie tratate ca simple valori în cea mai mare parte din cazuri. Aceasta este, de altfel, și cea mai eficientă tehnică, deoarece nu există overhead-ul reprezentat de lucrul cu referințe. Totuși, dacă vreți să folosiți aceste valori ca și cum ar fi referințe, ele pot fi temporar folosite ca variabile tip referință, printr-o operație de *boxing*.

Boxing

Expresii de tipuri valoare pot fi convertite către valori de tipul **object** și înapoi. Când o variabilă tip valoare trebuie convertită la tipul *object*, se alocă un obiect pentru a reține valoarea și această valoare este copiată în el. Procesul este cunoscut și ca *boxing*.

```
int p = 123;
object box;
box = p;           // Boxing (implicit)
box = (object) p;  // Boxing (explicit)
```

Operația de *boxing* poate fi făcută implicit sau explicit, folosind un cast. *Boxing* se folosește, de obicei, când o variabilă tip valoare este trimisă ca parametru unei metode ce așteaptă un object.

Unboxing

Când o valoare a unui obiect este convertită înapoi la o variabilă tip valoare, se copiază datele din obiect în locația corectă. Acest proces se numește *unboxing*

```
p = (int) box;
```

Această operație se face numai cu operator de cast explicit. Dacă valoarea stocată în obiect nu este de tipul la care încearcă să se facă cast, cast-ul va arunca o excepție **InvalidCastException**.

Apelarea metodelor clasei Object pentru tipuri valoare

Datorită faptului că operația de *boxing* se poate face implicit, puteți apela metode ale clasei Object pentru orice variabilă sau expresie, inclusiv pentru tipurile valoare.

```
static void Show(object o)
{
    Console.WriteLine(o.ToString( ));
}
Show(42);
```

Codul va funcționa pentru că valorii 42 i se face *boxing*, iar pentru obiectul rezultat se aplică metoda ToString.

Overview

- ▶ Tipuri referință
- ▶ Variabile de tip referință
- ▶ Tipul *Object*
- ▶ Boxing și Unboxing



27

Întrebări:

- Explicați cum este alocată memoria și eliberată pentru o variabilă tip referință
- Ce valoare specială este folosită pentru a indica faptul că o variabilă tip referință nu referențiază nici un obiect?
- Care sunt principalele caracteristici ale clasei *String*?
- Care este tipul de bază pentru toate clasele?