

СЪДЪРЖАНИЕ

- I. Обща система от типове (Common Type System).
- II. Йерархията на типовете в .NET.
- III. Стойностни и референтни типове.
- IV. Опаковане и разопаковане на обекти.

I. ОБЩА СИСТЕМА ОТ ТИПОВЕ (COMMON TYPE SYSTEM)

6

CLR (Common Language Runtime - виртуална машина, която контролирано изпълнява код написан за .NET платформата) поддържа много езици за програмиране. **За да се осигури съвместимост на данните между различните езици е разработена общата система от типове (Common Type System – CTS).** CTS дефинира поддържаните от CLR типове данни и операциите над тях.

Всички .NET езици използват типовете от CTS. За всеки тип в даден .NET език има някакво съответствие в CTS, макар че понякога това съответствие не е директно. Обратното не е вярно – съществуват CTS типове, които не се поддържат от някои .NET езици.

CTS е обектно-ориентирана система

По идея **всички езици в .NET Framework са обектно-ориентирани.** Common Type System също се придържа към идеите на обектно-ориентираното програмиране (ООП) и по тази причина описва освен стандартните типове (числа, символи, низове, структури, масиви) и някои типове данни свързани с ООП (например класове и интерфейси).

I. ОБЩА СИСТЕМА ОТ ТИПОВЕ (COMMON TYPE SYSTEM)

6

!!! CTS описва следните .NET типове:

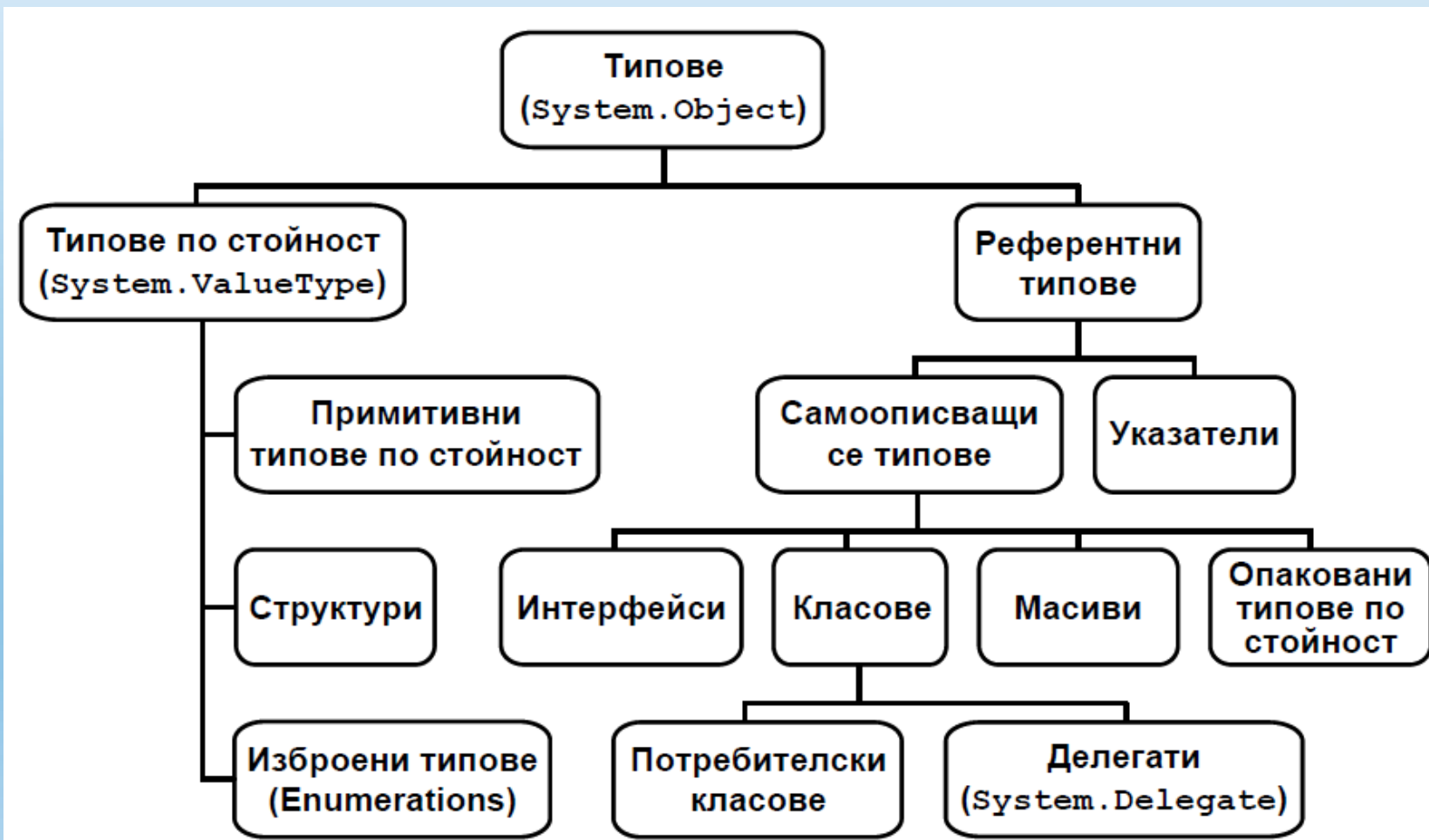
- примитивни типове (int, float, bool, char, ...)
- изброени типове (enums)
- класове (classes)
- структури (structs)
- интерфейси (interfaces)
- делегати (delegates)
- масиви (arrays)
- указатели (pointers)

Езикът C# и другите .NET езици използват CTS типовете и им съпоставят запазени думи съгласно своя синтаксис. Например типът System.Int32 от CTS съответства на типа int в C#, а типът System.String – на типа string.

II. ЙЕРАРХИЯТА НА ТИПОВЕТЕ В .NET

6

CTS дефинира строга йерархия на типовете данни, които се поддържат в .NET Framework:



В основата на йерархията стои системният тип `System.Object`. Той е общ предшественик (базов тип) за всички останали типове в CTS. Неговите преки наследници са стойностните и референтните типове.

Стойностните типове биват примитивни (`int`, `float`, `bool` и др.), структури (`struct` в C#) и изброени типове (`enum` в C#).

Референтните типове са всички останали — указателите, класовете, интерфейсите, делегатите, масивите и опакованите стойностни типове.

Типът `System.Object`

В CTS всички типове наследяват системния тип `System.Object`. Не правят изключение дори примитивните типове (`int`, `float`, `char`, ...) и масивите. **Всеки тип е наследник на `System.Object` и имплементира методите, включени в него.** Като резултат значително се улеснява работата с типове, защото променлива от произволен тип може да се присвои на променлива от базовия тип `System.Object` (`object` в C#).

Стойностни типове (value types)

Стойностни типове (типове по стойност) са повечето примитивни типове (int, float, bool, char и др.), структурите (struct в C#) и изброените типове (enum в C#).

Стойностните типове директно съдържат стойността си и се съхраняват физически в работния стек за изпълнение на програмата. Те не могат да приемат стойност null, защото реално не са указатели.

Стойностните типове и паметта - Стойностните типове заемат необходимата им памет в стека в момента на декларирането им и я освобождават в момента на излизане от обхват (при достигане на края на програмния блок, в който са декларирани).

Стойностните типове наследяват System.ValueType - CLR се грижи всички стойностни типове да наследяват системния тип System.ValueType. Всички типове, които не наследяват ValueType са референтни типове, т.е. реално са указатели към динамичната памет (адреси в паметта).



Референтни типове (reference types)

Референтни типове (типове по референция) са **указателите, класовете, интерфейсите, делегатите, масивите и опакованите стойностни типове.**

Физически референтните типове представляват указател към стойност в динамичната памет, но за CLR те не са обикновени указатели, а специални **типово-обезопасени указатели**. Това означава, че CLR не допуска на един референтен тип да се присвои стойност от друг референтен тип, който не е съвместим с него (т.е. не е същия тип или негов наследник). В резултат на това в .NET езиците грешките от неправилна работа с типове са силно намалени.

Референтните типове и паметта - Всички референтни типове се съхраняват в динамичната памет (т. нар. managed heap), която се контролира от системата за почистване на паметта (garbage collector). Динамичната памет е специално място от паметта, заделено от CLR за съхранение на данни, които се създават динамично по време на изпълнението на програмата. Такива данни са инстанциите на всички референтни типове.

III. СТОЙНОСТНИ И РЕФЕРЕНТНИ ТИПОВЕ

6

Когато инстанция на референтен тип престане да бъде необходима на програмата, тя се унищожава от системата за почистване на паметта (т. нар. garbage collector).

Когато създадем обект от референтен тип с оператора **new**, CLR заделя място в динамичната памет, където ще стоят данните и един указател в стека, който съдържа адреса на заделеното място. Веднага след това заделената памет се занулява

Ако референтен тип (например клас) съдържа член-данни от стойностен тип, те се съхраняват в динамичната памет. Ако референтен тип съдържа член-данни от референтен тип, в динамичната памет се заделят указатели (референции) за тях, а техните стойности (ако не са null) също се заделят също в динамичната памет, но като отделни обекти.

III. СТОЙНОСТНИ И РЕФЕРЕНТНИ ТИПОВЕ



Стойностни срещу референтни типове:

Стойностните и референтните типове в .NET Framework се различават съществено. Стойностните типове се разполагат в стека за изпълнение на програмата, докато референтните типове указатели към динамичната памет, където се съдържа самата им стойност.

Някои по-съществени разлики между тях:

- Стойностните типове наследяват системния тип `System.ValueType`, а референтните наследяват директно `System.Object`.
- При създаване на променлива от стойностен тип тя се заделя в стека, а при референтните типове – в динамичната памет.
- При присвояване на стойностни типове се копира самата им стойност, а при референтни типове – само референцията (указателя).

III. СТОЙНОСТНИ И РЕФЕРЕНТНИ ТИПОВЕ

- При предаване на променлива от стойностен тип като параметър на метод, се предава копие на стойността ѝ, а при референтните типове се предава копие на референцията, т.е. самата стойност не се ко-пира. В резултат, ако даден метод променя стойностен входен пара-метър, промените се губят при излизане от метода, а ако входният параметър е референтен, те се запазват.
- Стойностните типове не могат да приемат стойност null, защото не са указатели, докато референтните могат.
- Стойностните типове се унищожават при излизане от обхват, докато референтните се унищожават от системата за почистване на паметта (garbage collector) в някой момент, в който се установи, че вече не са необходими за работата на програмата.
- Променливи от стойностен тип могат да се съхраняват в променливи от референтен тип чрез т.нар. "опаковане" (boxing).

III. СТОЙНОСТНИ И РЕФЕРЕНТНИ ТИПОВЕ



Стойностни и референтни типове – пример

```
// SomeClass is reference type
class SomeClass
{
    public int mValue;
}

// SomeStruct is value type
struct SomeStruct
{
    public int mValue;
}

class TestValueAndReferenceTypes
{
    static void Main()
    {
        SomeClass class1 = new SomeClass();
        class1.mValue = 100;
        SomeClass class2 = class1; // Копира се референцията
        class2.mValue = 200; // Променя се и class1.mValue
        Console.WriteLine(class1.mValue); // Отпечатва се 200

        SomeStruct struct1 = new SomeStruct();
        struct1.mValue = 100;
        SomeStruct struct2 = struct1; // Копира се стойността
        struct2.mValue = 200; // Променя се копираната стойност
        Console.WriteLine(struct1.mValue); // Отпечатва се 100
    }
}
```

След като се изпълни примерът, се получава следния резултат:

```
C:\Examples\TypesExample\bin\Debug\TypesExample.exe"
200
100
Press any key to continue.
```

III. СТОЙНОСТНИ И РЕФЕРЕНТНИ ТИПОВЕ

6

Защита от неинициализирани променливи

Когато декларираме променлива в кода, C# компилаторът ни задължава да ѝ зададем стойност преди първото ѝ използване. Ако се опитаме да използваме неинициализирана променлива (независимо дали е от стойностен или референтен тип), C# компилаторът дава грешка и отказва да компилира кода. Ето един пример:

```
2 int someVariable;  
3 Console.WriteLine(someVariable);  
4
```

Error List

Entire Solution | 1 Error | 0 Warnings | 0 Messages

	Code	Description
✖	CS0165	Use of unassigned local variable 'someVariable'

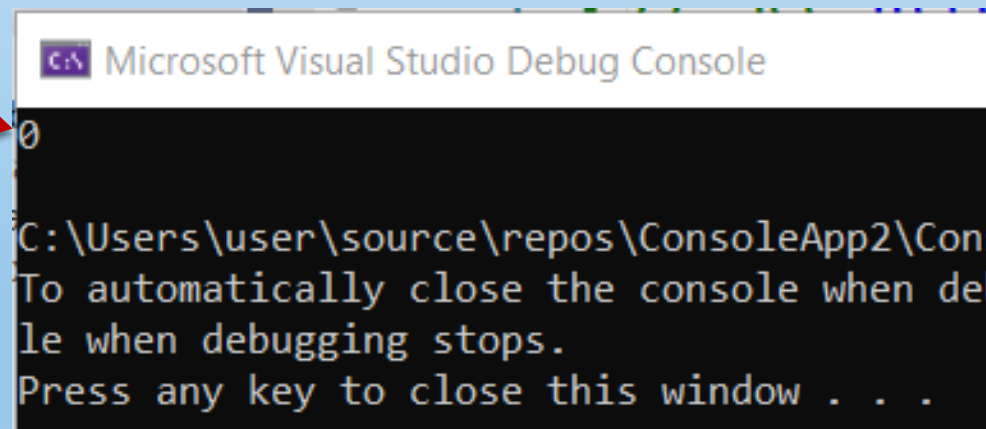
III. СТОЙНОСТНИ И РЕФЕРЕНТНИ ТИПОВЕ

6

Автоматична инициализация на променливите

При създаване на обект от даден тип с оператора `new` CLR автоматично инициализира декларираната променлива с неутрална (нулева) стойност. Ето един пример:

```
3  int i = new int();  
4  Console.WriteLine(i);
```



```
Microsoft Visual Studio Debug Console  
0  
  
C:\Users\user\source\repos\ConsoleApp2\ConsoleApp2\Program.cs:4: Console.WriteLine(i);  
To automatically close the console when debugging stops,  
press any key to close this window . . .
```

III. СТОЙНОСТНИ И РЕФЕРЕНТНИ ТИПОВЕ



Типът System.Object

Типът System.Object е базов за всички типове в .NET Framework. Както референтните, така и стойностните типове произлизат от System.Object или от негов наследник. Това улеснява програмиста и в много ситуации му спестява писане на излишен код.

В .NET Framework можем да напишем следния код:

```
6 | string s = 5.ToString();
```

Този код извиква виртуалния метод ToString() от класа System.Object. Това е възможно, защото числото 5 е инстанция на типа System.Int32, който е наследник на System.Object.

Понеже всички типове са съвместими със System.Object, защото са негови наследници, можем на инстанция на System.Object да присвояваме както референтни, така и стойностни типове:

```
object obj = 5;  
object obj2 = new SomeClass();
```


III. СТОЙНОСТНИ И РЕФЕРЕНТНИ ТИПОВЕ



Защо стойностните типове наследяват референтния тип System.Object?

Архитектите на .NET Framework по изкуствен начин са направили съвместими всички стойностни типове с референтния тип System.Object. **За удобство в CLR всички стойностни типове могат да се преобразуват към референтни чрез операцията "опаковане"**. Опаковането и обратната му операция "разопаковане" преобразуват стойностни типове в опаковани стойностни типове и обратното. При опаковане стойностните типове се копират в динамичната памет и се получава указател (референция) към тях. При разопаковане стойността от динамичната памет, сочена от съответната референция, се копира в стека.

Потребителските типове скрито наследяват System.Object

При дефиниране на какъвто и да е тип, скрито от нас се наследява System.Object. Например структурата е наследник на System.Object, макар това да не се вижда непосредствено от декларацията ѝ.

```
struct Point
{
    int x, y;
}
```

Методите на System.Object

Като базов тип за всички .NET типове System.Object дефинира обща за всички тях функционалност. Тази функционалност се реализира в няколко метода, някои от които са виртуални:

- `bool Equals(object)` – виртуален метод, който сравнява текущия обект с друг обект. Методът има и статична версия `Equals(object, object)`, която сравнява два обекта, подадени като параметри. Обектите се сравняват не по адрес, а по съдържание. Методът често пъти се припокрива, за да се даде възможност за сравнение на потребителски обекти.

- `string ToString()` – виртуален метод, който представя обекта във вид на символен низ. Имплементацията по подразбиране на `ToString()` отпечатва името самия тип.

- `int GetHashCode()` – виртуален метод за изчисляване на хеш-код. Използва се при реализацията на някои структури от данни, например хеш-таблицы.
- `Finalize()` – виртуален метод за имплементиране на почистващи операции при унищожаване на обект.
- `Type GetType()` – връща метаданни за типа на обекта във вид на инстанция на `System.Type`. Имплементиран е вътрешно от CLR.
- `object MemberwiseClone()` – копира двоичното представяне на обекта в нов обект, т. е. извършва плитко копиране. При референтни типове създава нова референция към същия обект. При стойностни типове копира стойността на подадения обект.
- `bool ReferenceEquals()` – сравнява два обекта по референция. При референтни типове се сравнява дали обектите сочат на едно и също място в динамичната памет. При стойностни типове връща `false`.

IV. ОПАКОВАНЕ (BOXING) И РАЗОПАКОВАНЕ (UNBOXING) НА СТОЙНОСТНИ ТИПОВЕ



Понякога се налага на референтен тип да се присвои обект от стойностен тип. Например може да се наложи в System.Object инстанция да се запише System.Int32 стойност. CLR позволява това благодарение на т. нар. "опакване" на стойностните типове (boxing). В .NET Framework стойностните типове могат да се използват без преобразуване навсякъде, където се изискват референтни типове. При нужда CLR опакова и разопакова стойностните типове автоматично. Това спестява дефинирането на обвиващи (wrapper) класове за примитивните типове, структурите и изброените типове, но разбира се, може да доведе и до някои проблеми.

Опаковане (boxing) на стойностни типове

Опаковането (boxing) е действие, което преобразува стойностен тип в референция към опакована стойност. **То се извършва, когато е необходимо да се преобразува стойностен тип към референтен тип**, например при преобразуване на Int32 към Object:

```
int i = 5;  
object obj = i; // i се опакова
```

IV. ОПАКОВАНЕ (BOXING) И РАЗОПАКОВАНЕ (UNBOXING) НА СТОЙНОСТНИ ТИПОВЕ



Всяка инстанция на стойностен тип може да бъде опакована чрез просто преобразуване до `System.Object`. Ако един тип е вече опакован, той не може да бъде опакован втори път и при преобразуване към `System.Object` си остава опакован само веднъж.

CLR извършва опаковането по следния начин:

1. Заделя динамична памет за създаване на копие на обекта от стойностния тип.
2. Копира съдържанието на стойностната променливата от стека в заделената динамична памет.
3. Връща референция към създадения обект в динамичната памет.

При опаковането в динамичната памет се записва информация, че референцията съдържа опакован обект и се запазва името на оригиналния стойностен тип.

IV. ОПАКОВАНЕ (BOXING) И РАЗОПАКОВАНЕ (UNBOXING) НА СТОЙНОСТНИ ТИПОВЕ



Разопаковане (unboxing) на опаковани типове

Разопаковането (unboxing) е процесът на извличане на опакована стойност от динамичната памет. Разопаковане се извършва при преобразуване на опакована стойност обратно към инстанция на стойностен тип, например при преобразуване на Object към Int32:

```
object obj = 5; // 5 се опакова  
int value = (int) obj; // стойността на obj се разопакова
```

CLR извършва разопаковането по следния начин:

1. Ако референцията е null се предизвиква `NullReferenceException`.
2. Ако референцията не сочи към валидна опакована стойност от съот-ветния тип, се предизвиква изключение `InvalidCastException`.
3. Ако референцията е валидна опакована стойност от правилния тип, стойността се извлича от динамичната памет и се записва в стека.

IV. ОПАКОВАНЕ (BOXING) И РАЗОПАКОВАНЕ (UNBOXING) НА СТОЙНОСТНИ ТИПОВЕ



Особености при опаковането и разопаковането

При използване на автоматично опаковане и разопаковане на стойности трябва да се имат предвид някои особености:

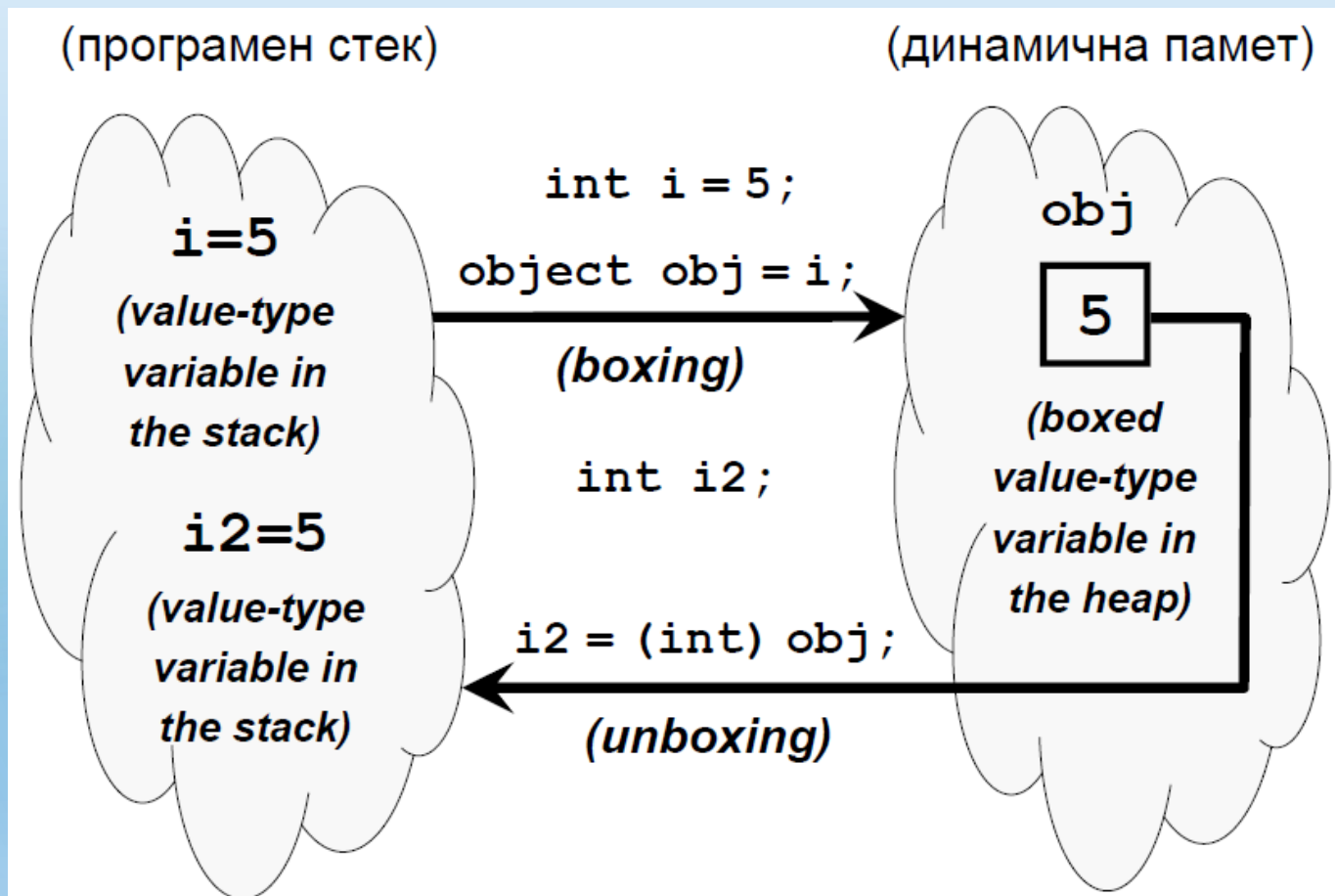
- Опаковането и разопаковането намаляват производителността. За оптимална производителност трябва да се намали броят на опакованите и разопакованите обекти.
- Опакованите типове са копия на оригиналните стойности, поради което, ако променяме оригиналния неопакван тип, опакованото копие не се променя.

Как работят опаковането и разопаковането?

```
int i = 5;  
object obj = i; // boxing  
  
int i2;  
i2 = (int) obj; // unboxing
```

IV. ОПАКОВАНЕ (BOXING) И РАЗОПАКОВАНЕ (UNBOXING) НА СТОЙНОСТНИ ТИПОВЕ

6



При опаковане стойността от стека се копира в динамичната памет, а при разопаковане стойността от динамичната памет се копира обратно в стека.

Опакованите стойности се държат като останалите референтни типове — разполагат се в динамичната памет, унищожават се от garbage collector, когато не са необходими на програмата, и при подаване като параметър при извикване на метод се пренасят по адрес.

IV. ОПАКОВАНЕ (BOXING) И РАЗОПАКОВАНЕ (UNBOXING) НА СТОЙНОСТНИ ТИПОВЕ



Пример за опаковане и разопаковане

```
using System;

class TestBoxingUnboxing
{
    static void Main()
    {
        int value1 = 1;
        object obj = value1; // извършва се опаковане

        value1 = 12345; // променя се само стойността в стека

        int value2 = (int)obj; // извършва се разопаковане
        Console.WriteLine(value2); // отпечатва се 1

        long value3 = (long) (int) obj; // разопаковане

        long value4 = (long) obj; // InvalidCastException
    }
}
```