**55. Генетични типове и наследяемост. Синтактично подменяне на генетичен тип. Обработка на генетични типове. Ограничители.**

Open & Closed types

Тип с генетични параметри се нарича ‘open type’ тъй като не допуска CLR да конструира инстанции директно ( както е и при интерфейсите) Когато кодът се обърне към генетичен тип, се подават реални параметри. Тогава типът се нарича вече ‘ closed type’ и за него се прави инстанция.

**1.Генетични типове и наследяемост:**

Това си е нормален тип и наследяемост е напълно допустима.

*internal sealed class Node<T> {*

*public T m\_data; public Node<T> m\_next;*

*public Node(T data) : this(data,nul) {}*

*public Node(T data, Node<T> next) {*

*m\_data = data; m\_next = next; } ….*

*}*

Използваме в производен тип:

*private static void SameDataLinkedList() {*

*Node<Char> head = new Node<Char>(‘C’);*

*head = new Node<Char>(‘B’, head);*

*head = new Node<Char>(‘A’, head);*

*}*

**2.Подменяне на генетични типове:**

Подменяне на генетични типове с цел удобство, е честа практика:

ако имаме: List<DateTime> dtl = new List<DateTime>();

да предефинираме: internal sealed class DateTimeList : List<DateTime> {}

И тогава можем да създадем списък от генетичен тип по традиционния начин: DateTimeList dtl = new DateTimeList();

**3.Обработка на генетични типове: *code explosion***

- При повикване на метод от генетичвен тип, JIT компилаторът прави заместването и създава ‘ native code’ за точно този метод с точно тези подменени параметри.

-CLR генерира native code за всеки метод/тип комбинация. Това води до ‘code explosion’.

-Ako впоследствие, метод се повика със същия тип аргумент, не се генерира повторен код.

-Еднократно се генерира и код в случаите, когато сргументите са от референтен тип. Напр: List<String> List<Stream> макар и аргументите всъщност да сочат съвсем различни неща.

При наследяване от негенетичен към генетичен интерфейс, трябва да се вътрешно пакетиране (преобразуване) на аргументите(boxing), Koeто е загуба на ресурс и бързо действие

Ето един стандартен в FCL интерфейс:

*public interface IEnumerator<T> : IDisposable, IEnumerator {*

*T Current { get;}*

*}*

Eто клас, който имплементира горния интерфейс над тип Point:

*internal sealed class Triangle : IEnumerator<Point> {*

*private Point[] m\_vetrices;*

*public Point Current { get {… }*

*}*

**4.Ограничители (в генетични типове)**

Чрез тях може да се ограничи броя на типовете, които могат да са заместители в аргументите на генетичен тип:

*public static T Min<T>(T o1, Y o2) where T : IComparable*

*{*

*if(o1.CompareTo(o2) < 0) return o1;*

*return o2;*

*}*