Generics

--

Узагальнення

Generics - Узагальнення

- Узагальнення – механізм CLR повторного використання коду
- Узагальнені типи і їх компоненти – конструкції, побудовані з використанням *параметрів*, які представляють типи в програмному коді
- Види узагальнених конструкцій:
  - узагальнені класи (типи-посилання)
  - узагальнені структури (типи-значення)
  - узагальнені інтерфейси
  - узагальнені делегати
  - узагальнені методи

Узагальнені типи і об'єкти-типи

- Узагальненим типам CLR ставить у відповідність відкриті об'єкти-типи, для яких не можуть створюватися екземпляри.
- Якщо кожному параметру узагальненя буде поставлено у відповідність конкретний тип, то при посиланні на відкритий тип він вважатиметься закритим (придатним для утворення екземплярів),
- CLR розміщує **статичні поля** в об'єкті-типі, тому кожен закритий тип має свої статичні поля;

`List<int>` і `List<string>` матимуть окремі статичні поля

GenericStack

Приклади узагальнених інтерфейсів

```
using System.Collections;
using System.Collections.Generic;
```

```
[SerializableAttribute]
public class List<T> : IList<T>, ICollection<T>, IEnumerable<T>,
                       IList, ICollection, IEnumerable
```

```
public interface IEnumerator<T> : IDisposable,IEnumerator
{
  T Current{get;}
}
```

```
class Triangle : IEnumerator<Point>
{
   . . .
   Point Current{get {. . .}}
}
```

Узагальнені делегати (приклади)

```
public delegate TReturn CallB < TReturn, TKey, TValue>
                                ( TKey key,TValue value);
```

FCL:

```
public delegate void Action<T>(T obj);

public delegate Int32 Comparison<T>(T x, T y);

public delegate TOutput Converter<TInput,TOutput>(TInput input);
```

Узагальнені методи

```
public class SClass{
    public void SMethod<T>(T t) {...}
}
...
var obj = new SClass();
obj.SMethod<int>(3);
```

```
public class SClass<T>{
    public void SMethod<X>(X x){...}
}
```

```
public class SClass<T>{
    public static T SomeMethod<X>(T t,X x) {...}
}
. . .
var number = SClass<int>.SomeMethod<string>(3,"AAA");
```

Виведення типів
Type inference

```
public static void Display(string s)
{
  Console.WriteLine(s);
}

public static void Display<T>(T o)
{
    Display(o.ToString());
}
....
Display("1234"); // Display(string);
Display(1234); // Display<T>(T);
Display<string>("1234"); // Display<T>(T);
```

Обмеження на тип аргумента -- Constraints

`where T: struct`	Тип аргумента повинен бути будь-яким **non-nullable** типом-значенням
`where T : class`	Тип аргумента повинен бути **non-nullable** типом-посиланням: довільний клас, інтерфейс, делегат чи масив
`where T : class ?`	Тип аргумента повинен бути **nullable** або **non-nullable** типом-посиланням: довільний клас, інтерфейс, делегат чи масив
`where T : new()`	Тип аргумента повинен мати public конструктор за замовчуванням; встановлюється останнім у списку обмежень
`where T : <base class name>`	Тип аргумента повинен бути **non-nullable** або вказаним типом, або типом, похідним від вказаного
`where T : <base class name> ?`	Тип аргумента повинен бути **nullable** або **non-nullable** або вказаним типом, або типом, похідним від вказаного
`where T : <interface name>`	Тип аргумента повинен бути або типом даного інтерфейсу, або **non-nullable** типом, що реалізує даний інтерфейс; можлива довільна кількість обмежень-інтерфейсів; Інтерфейс-обмеження може теж бути узагальненням
`where T : <interface name> ?`	Тип аргумента повинен бути **nullable** або **non-nullable** референсним типом, що реалізує даний інтерфейс; або типом-значенням
`where T : U`	Тип аргумента T повинен бути або типом U, або типом, похідним від U -- явне обмеження типу

Nullable reference types

- Nullable aware context
  - Nullable reference types are available in code that has opted in to a nullable aware context.
  - A nullable context is controlled at the project level using build settings, or in code using pragmas.

- In a nullable aware context:
  - A variable of a reference type T must be initialized with **non-null**, and may never be assigned a value that may be **null**.
  - A variable of a reference type T? may be initialized with **null** or assigned **null**, but is required to be checked against **null** before **de-referencing**.
  - A variable **m** of type T? is considered to be non-null when you apply the null-forgiving operator, as in **m!**

Nullable value types

- A nullable value type **T**? represents all values of its underlying value type **T** and an additional **null** value.

  For example (when database field is undefined or missing) :
  for variable bool? v can be assigned true, false, or null

- Any nullable value type is an instance of the generic System.Nullable<T> structure. Nullable<T> or T? -- interchangeable forms

Обмеження на тип

- Type parameter bounds a type set

```
public class PrimaryStream<T> where T: Stream
{
    public void M( T stream )
    {
        stream.Close();
    }
}
```

- Constraints may be applied to multiple parameters, and multiple constraints may be applied to a single parameter

```
class Base { . . . }
class Test<T, U>
       where U : struct
       where T : Base, new()
{. . . }
```

- Type parameter V is used as a type constraint for T

```
public class SampleClass<T, U, V> where T : V
{ . . .}
```

- When applying the where T : class constraint, avoid the == and != operators on the type parameter because these operators will test for reference identity only, not for value equality.

Обмеження конструктора

- вказує компілятору, що аргумент-тип буде неабстрактного типу, який реалізує відкритий конструктор без параметрів

```
class Node<K,T> where T : new()
{
    public K Key;
    public T Item;
    public Node<K,T> NextNode;

    public Node()
    {
        Key = default(K);
        Item = new T();
        NextNode = null;
    }
}
```

```
public class LinkedList<K,T> where K : IComparable<K>,new()
{...}
```

Приведення типу параметра

- допускається лише тоді, коли змінна приводиться до типу, який дозволений обмеженням

```
private static void Clm<T>(T obj)
{
    Int32 x = (Int32) (Object) obj;
    String s = (String) (Object) obj;
}
```

- приведення до типу посилання оператором **as**

```
private static void Clm<T>(T obj)
{
          String s=obj  as String;
}
```

Generics in the Run Time

- The metadata of the generic identifies it for MSIL-compiler as having type parameters
- Specialized generic types are created one time for each unique **value type** that is used as a parameter
- The first time a generic type is constructed with any **reference type**, runtime creates a specialized generic with **object references** substituted for the parameters in the MSIL