Узагальнення (Generics) - механізм CLR повторного використання коду

Клакович Л.М.

Узагальння в C#

- Узагальнення – концепція параметризованих типів в .NET
- Види узагальнених конструкцій
- Приклади узагальнених інтерфейсів, методів, делегатів
- Виведення типів в узагальненнях
- Обмеження (Constraints) в узагальненнях
- Колекції FCL

Приклад узагальненого типу

```
// Оголошення узагальненого класу
public class GenericList<T>
{
    void Add(T input) { }
}
class TestGenericList
{
    private class ExampleClass { }
    static void Main()
    {
        // оголошення списку елементів типу int
        GenericList<int> list1 = new GenericList<int>();

        // Оголошення списку  елементів типу string
        GenericList<string> list2 = new GenericList<string>();

        // Оголошення списку  елементів типу ExampleClass
        GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();
    }
}
```

Узагальнення – параметризовані типи в .NET

- Узагальнення були додані у версію 2.0 C#
- Узагальнення в .NET Framework представляють концепцію параметризованих типів - аналог шаблонів (templates) в C++
- Дозволяють розробляти параметризовані класи і методи, відкладаючи специфікацію типу до моменту оголошення та утворення в коді.
- В основному узагальння використовуються з колекціями - простір імен System.Collections.Generic містить узагальнені колекції

Види узагальнених конструкцій:

- **узагальнені класи** та **структури**
- **узагальнені інтерфейси**
- **узагальнені делегати**
- **узагальнені методи**

Для чого потрібні узагальнення?

```
class Account
    {
        public int Id { get; set; }
        public int Sum { get; set; }
    }
```

- Клас представляє банківський рахунок
- Id може бути цілим числом або рядком (string)
- Можна надати Id тип object – **чому ні?**

Для чого потрібні узагальнення?

```
class Account<T>
    {
        public T Id { get; set; }
        public int Sum { get; set; }
    }
```

- Тепер це узагальнений клас
- Id може бути цілим числом або рядком (string) або іншим типом

```
Account<int> account1 = new Account<int> { Sum = 5000 };
account1.Id = 2;
int id1 = account1.Id;

Account<string> account2 = new Account<string> { Sum = 4000 };
account2.Id = "4356";
string id2 = account2.Id;
```

- Значення за замовчуванням

```
class Account<T>
   {
        T id = default(T);
   }
```

- Статичні поля узагальнених класів

```
class Account<T>
   {
       public static T session;

       public T Id { get; set; }
       public int Sum { get; set; }
   }
```

При типізації узагальненого класу певним типом буде створюватися свій набір статичних членів.

```
Account<int> account1 = new Account<int> { Sum = 5000 };
Account<int>.session = 5436;

Account<string> account2 = new Account<string> { Sum = 4000 };
Account<string>.session = "45245";
```

➢ Використання декількох універсальних параметрів

```
class Transaction<U, V>
    {
        public U FromAccount { get; set; }
        public U ToAccount { get; set; }
        public V Code { get; set; }
        public int Sum { get; set; }
    }м 5 ь ьл
```

```
Account<int> acc1 = new Account<int> { Id = 1857, Sum = 4500 };
Account<int> acc2 = new Account<int> { Id = 3453, Sum = 5000 };

Transaction<Account<int>, string> transaction1 =
                        new Transaction<Account<int>, string>
            {
                FromAccount = acc1,
                ToAccount = acc2,
                Code = "45478758",
                Sum = 900
            };
```

Узагальнені методи

```
public class MyClass<T>
{
      public void MyMethod<X>(X x){...}
}
```

```
public class MyClass
{
      public void MyMethod<T>(T t)    {...}
}
...
MyClass obj = new MyClass();
obj.MyMethod<int>(3);
```

```
public class MyClass<T>
{
  public static T SomeMethod<X>(T t,X x)    {...}
}

int number = MyClass<int>.SomeMethod<string>(3,"AAA");
```

Узагальнені методи. Метод Swap()

```
private static void Swap<T>(ref T o1, ref T o2)
{
            T temp = o1;
            o1 = o2;
            o2 = temp;
}
```

```
public void CallingSwap()
{
             Int32 n1 = 1, n2 = 2;
             Swap<Int32>(ref n1, ref n2);

             String s1 =“Ai”, s2 =“Kr”;
             Swap<String>(ref s1, ref s2);
}
```

Приклади узагальнених інтерфейсів

- **Тип реалізує узагальнений інтерфейс, задаючи аргументи-типи:**

```
class Triangle : IEnumerator<Point>
{
        private Point[ ] m_ver;
//…
}
```

- **не визначаючи аргументи-типи, тип стає узагальненим:**

```
class ArrayEnumerator<T> : IEnumerator<T>
{
        private T[ ] m_array;
//…
}
```

Приклади узагальнених інтерфейсів

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

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

Обмеження в узагальненнях

```
// працює з будь-яким типом
public bool MethodTakingAnyType<T>(T o)
{
            T temp = o;
            Console.WriteLine(o.ToString());
            Boolean b = temp.Equals(o);
            return b;
}
```

```
// ?
public T Min<T>(T o1, T o2)
{
            if (o1.CompareTo(o2) < 0) return o1;
            return o2;
}
```

Обмеження в узагальненнях

- Обмеження дозволяють звузити перелік типів, які можна передати в узагальненому аргументі.
- Встановлюються через **where T:**

```
public T Min<T>(T o1, T o2) where T : IComparable<T>
{
            if (o1.CompareTo(o2) < 0) return o1;
            return o2;
}

Object o1 =“Jeff”, o2=“Richter”;
Object oMin = Min<Object>(o1, o2);      // error
```

Типи обмежень

`where T: struct`	Тип аргумента повинен бути типом-значенням. Будь-який тип-значення, крім Nullable.
`where T : class`	Тип аргумента повинен бути типом-посиланням, включаючи будь-який 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 або вказаним типом, або типом, похідним від вказаного.
`where T : <interface name>`	Тип аргумента повинен бути або типом даного інтерфейсу, або типом, що реалізує даний інтерфейс. Можна визначати декілька обмежень-інтерфейсів. Інтерфейс-обмеження може теж бути узагальненням
`where T : U`	Тип аргумента T повинен бути або типом U, або типом, похідним від U. Явне обмеження типу.

Основні обмеження

- Не можна використовувати: Object, Array, Delegate, ValueType, Enum, void.
- В параметрі-типі можна задати не більше одного основного обмеження
- Основні обмеження: **class** і **struct**

```
public class MyClass<T> where T : class
{
    public void M(){ T temp=null; }
}
```

```
public class YouClass<T> where T : struct
{
          public T Factory(){return new T();}
}
```

- **Обмеження типом:**

```
public class MyOtherClass{...}
public class MyClass<T> where T : MyOtherClass {...}
MyClass<MyOtherClass> obj =
                          new MyClass<MyOtherClass>();
```

- **Кожен узагальнений параметр може мати власне обмеження:**

```
public class LinkedList<K,T> where K : IComparable<K>
                             where T : ICloneable
 {...}
```

- Обмеження узагальненого параметра інтерфейсом

```
public interface IMyInterface{...}
public class MyClass<T> where T : IMyInterface{...}
```

```
public class LinkedList<K,T> where K : MyBaseClass,
                                       IComparable<K>,
                                       IMyInterface
```

- При обмеженні узагальненого параметра іншим параметром необхідно існування певних відношень між ними

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

- **Обмеження конструктора** задається тільки одне. Вказує компілятору, що аргумент-тип буде неабстрактного типу, що 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()
{...}
```