Обобщения (generics)

- ✔ Обобщения – это классы, структуры, интерфейсы и методы, в которых некоторые типы сами являются параметрами. Эти типы перечисляются в списке параметров-типов(placeholders).
- ✔ Синтаксически обобщения похожи на шаблоны в неуправляемом C++. Поддерживаются только версиями .NET Framework 2.0 и выше.
- ✔ Определение обобщенного типа задает шаблон, в котором некоторые используемые типы являются параметрами и будут конкретизированы при создании объектов обобщенного типа.
- ✔ Пример определения обобщенного типа

```
public class MyList <T> {
    …
}
```

- ✔ Создание объектов обобщенного типа

```
MyList <string> ms = new MyList <string>() ;
MyList <StringBuilder> ms = new MyList <StringBuilder>() ;
```

Зачем нужны обобщения (generics)?

- ✔ В библиотеке классов NET Framework 1.0 -1.1 классы-коллекции определены как состоящие из элементов типа Object.
- ✔ Две проблемы:
  - операция упаковки (boxing) при добавлении в коллекцию элемента, имеющего тип-значение.
  - невозможность проверить безопасность типа на стадии компиляции.

Обобщения и CLR

- ✔ Синтаксически обобщения похожи на шаблоны в неуправляемом C++, но компилируются в native-код в процессе выполнения приложения.
- ✔Обобщения (generics) поддерживаются IL и CLR. Как и для обычных типов, код обобщенного типа компилируется в IL и информация об обобщенных параметрах размещается в метаданных.
- ✔ В процессе компиляции IL-кода в команды процессора типовым параметрам присваиваются значения.
- ✔ В случае, когда обобщенный параметр получает как значение ссылочный тип, используется один и тот же native код.
- ✔ Для каждого типа-значения для обобщенного параметра , JIT-компилятор создает новый код, но не дублирует уже созданный.
- ✔ Обобщения дают возможность создавать эффективный, безопасный с точки зрения использования типов код.
- ✔ MSDN. Статья Juval Lowy “An Introduction to C# Generics”.

Обобщенные методы (generic methods)

- ✔ Обобщенный метод может использовать тип-параметр как тип возвращаемого значения или как тип одного из формальных параметров.
- ✔ Обобщенный метод можно объявить как в обычном классе (структуре), так и в обобщенном классе (структуре).
- ✔ Обобщенный метод - это метод со своим списком обобщенных параметров.
- ✔ В приведенном примере только метод G() считается обобщенным:

```
class MyClass
{    T G<T>(T arg) {...}
}
class MyGenericClass<T>
{    T M(T arg) {...}
}
```

Ограничения (constraints)

- ✔ Можно указать ограничения для типов, которые могут быть значениями обобщенных типов-параметров.
- ✔ .NET Framework 2.0 и выше поддерживают следующие ограничения:
  - тип должен реализовывать определенные интерфейсы;
  - тип должен иметь определенный базовый класс;
  - тип должен иметь конструктор без параметров (default ctor);
  - тип должен быть ссылочным типом или типом-значением.
- ✔ Информация об ограничениях хранится в метаданных и используется JIT-компилятором.

Ограничения (constraints) -2

- ✔ Ограничения задаются с помощью ключевого слова where.
- ✔ В следующем примере тип, который может быть значением типового параметра K, должен реализовывать интерфейс IComparable<K> и иметь базовый класс MyBaseClass, а тип T – реализовывать интерфейсы IDisposable и ICloneable:

```
class MyGenericClass <K,T>    where K : MyBaseClass, IComparable<K>
                    where T : ICloneable, IDisposable
{...}
```

- ✔ В списке ограничений можно указать только один базовый класс. Базовый класс должен быть указан первым (до интерфейсов).
- ✔ Обобщенный параметр можно использовать в ограничениях для другого параметра:

```
class MyGenericClass <K,T>  where K : T, IComparable
{...}
```

Ограничения (constraints) -3

✔ В следующем примере тип K должен иметь открытый конструктор без параметров:

```
class MyGenericClass <K,T>    where K : MyBaseClass, new()
                      where T : ICloneable, IDisposable
{...}
```

✔ В следующем примере тип, который может быть значением параметра K, должен быть ссылочным типом, а для типа T – типом-значением:

```
class MyGenericClass <K,T>    where K : class, IDisposable
                  where T : struct, IComparable
{...}
```

Обобщения и наследование

✔ Если базовый класс является обобщенным типом, то при определении производного класса

- или обобщенный параметр базового класса должен быть обобщенным параметром производного класса;
- или необходимо использовать конкретное значение для обобщенного типа.

```
public class Base<T> { . . . }
public class Derived : Base <string> { . . . }
public class GenericDerived <T> : Base <T> { . . . }
```

✔ Если для обобщенных параметров базового класса указаны ограничения, их необходимо повторить при объявлении производного класса.

```
public class Base<T> where T : IDisposable { . . . }
public class Derived<T> : Base<T> where T : IDisposable
{ . . . }
```

Обобщения и неявное приведение типов

- ✔ Допускается неявное приведение обобщенного типа
  - к типу object;
  - к типу, возможность приведения к которому следует из ограничений.

```
public class MyClass <T> where T: MyBaseClass, IMyInterface
{
  void MyMethod (T t)
    { object obj1 = t;
      MyBaseClass obj2 = t;
      IMyInterface obj3 = t;
      . . .
    }
 . . .
}
```

Обобщения и явное приведение типов

- ✔ Допускается явное приведение обобщенного типа к любому интерфейсу.
- ✔ Не разрешено явное приведение к классу.

```
public class MyClass <T> where T: MyBaseClass, IMyInterface
{
  void MyMethod (T t)
    { IComparable obj1 = (IComparable)t;
      // string obj2 = (string)t;
     . . .
    }
 . . .
}
```

- ✔ Можно выполнить явное приведение T-> object -> string. При выполнении может быть брошено исключение.

Обобщенные методы (generic methods)

- ✔ Метод в обобщенном классе может иметь свои обобщенные параметры.
- ✔ Свойства и индексаторы могут использовать только обобщенные параметры класса (структуры).
- ✔ Если метод определяет свой обобщенный параметр, для него можно задать ограничения.
- ✔ При вызове обобщенного метода можно не указывать явно тип обобщенного параметра, если компилятор может определить его по типу фактического параметра метода.

```
class MyClass
{ T Method1<T>(T arg)
    { return default(T);  }
  T Method2<T>()
    { return default(T);  }
}
```

```
static void Main()
{  MyClass mc = new MyClass();
   mc.Method1(123);
   mc.Method1("abc");
   mc.Method2<string>();
}
```

Оператор default

Оператор default(T) возвращает значение по умолчанию для типа:

- null - для ссылочных типов;
- 0 - для отдельных типов-значений (int, double,..);
- для структур (struct) всем полям присваиваются значения 0 или null, в зависимости от того, какой тип имеет поле – тип-значение или ссылочный тип.

```
// При попытке выбрать элемент из пустого стека метод Pop() не
// бросает исключение, а возвращает значение по умолчаню – null
// для стека из строк и 0 для стека из целых чисел.
public class Stack<T>
{  int m_StackPointer = 0;
   […….код C# ]
   public T Pop()
    {  […….код C# ]
       if(m_StackPointer == 0) return default(T);
       }
}
```

Типы, принимающие значение null

- В приложениях, работающих с базами данных, переменные могут находиться в “неопределенном” состоянии. Для ссылочных типов – это значение null.
- В .NET Framework 2.0 и выше добавлена поддержка null для типов-значений с помощью обобщенного типа System.Nullable<T>.

```
public struct Nullable<T>  where T : struct
{   public Nullable (T value) ;
    public bool HasValue { get; }
    public T Value { get; }
    public T GetValueOrDefault ();
    public T GetValueOrDefault ( T defaultValue );
    public static explicit operator T (Nullable<T> value);
    public static implicit operator Nullable<T> (T value);
}
```

- Два способа объявления nullable переменной:
  - `System.Nullable<T> variable;`
  - `T? variable;`

System.Nullable<bool>

- ✔ Для System.Nullable<bool> поддерживается троичная логика:

x	y	x&y	x|y
true	true	true	true
true	false	false	true
true	null	null	true
false	true	false	true
false	false	false	false
false	null	false	null
null	true	null	true
null	false	false	null
null	null	null	null

Статические обобщенные методы в System.Array

- создание read-only оболочки для массива ;
- преобразование массива одного типа в массив другого типа;
- бинарный поиск (4 перегрузки);
- поиск элемента, совпадающего с заданным объектом;
- поиск элементов массива, для которых выполняются условия, определенные в предикате;
- сортировка массива (9 перегрузок).

```
// Изменение числа элементов в массиве
public static void Resize<T> ( ref T[] array, int newSize) ;

// Создание  read-only оболочки для массива
public static ReadOnlyCollection<T> AsReadOnly<T>( T[] array );

// Преобразование массива с элементами типа TInput в массив
// с элементами типа  TOutput
public static TOutput[] ConvertAll<TInput,TOutput> ( TInput[] array,
            Converter<TInput,TOutput> converter ) ;
```

Обобщенный интерфейс IComparable<T>

✔ Интерфейс IComparable и обобщенный интерфейс IComparable<T> определены в пространстве имен System

```
public interface IComparable
{int CompareTo ( Object obj );
}
public interface IComparable<T>
{int CompareTo( T other );
}
```

✔ Интерфейс IComparable используется при сортировке ArrayList. Обобщенный интерфейс IComparable<T> используется при сортировке массивов в статических методах Sort из класса System.Array.

Обобщенные интерфейсы в BCL

- ✔ Следующие обобщенные интерфейсы определены в пространстве имен System.Collections.Generic.

```
public interface IEnumerable<T> : IEnumerable
{   IEnumerator<T> GetEnumerator ();}

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

public interface IComparer<T>
{   int Compare ( T x, T y );}

public interface IEqualityComparer<T>
{   bool Equals ( T x, T y );
    int GetHashCode (T obj);
}
```