Обобщенные типы

Обобщенные методы

Чтобы написать обобщенный метод, сначала нужно написать обычный, а потом заменить конкретный тип параметром.

```
static int indexOf(char[] m, char k)
{
    for (int i = 0; i < m.Length; i++)
        if (m[i] == k)
            return i;
    return -1;
}
```

```
static int indexOf<T>(T[] m, T k)
{
    for (int i = 0; i < m.Length; i++)
        if (m[i].Equals(k))
            return i;
    return -1;
}
```

```
int n = indexOf<int>(list, 35);

int n = indexOf(list, 35);
```

Вывод типа – при вызове обобщенного метода компилятор может сам конкретизировать тип.

Ограничения параметрического типа

```
static int indexOf<L, T>(L m, T k) where L: IList<T>
{
    for (int i = 0; i < m.Count; i++)
         if (m[i].Equals(k))
             return i;
    return -1;
}
```

Специфика параметров должна быть описана в ограничениях (ключевое слово where)

Форма ограничений

where T: struct

where T: class

where T: InterfaceName

where T: ClassName

where T: new()

Обобщенные классы

```
public class Rect<T>
{
    public T Width { set; get; }
    public T Height { set; get; }

    public double Square()
    {
        return Convert.ToDouble(Width) *
               Convert.ToDouble(Height);
    }
}
```

Иным способом умножение объектов типа T можно организовать при помощи интерфейса с методом умножения.

Работа с классом Rect<T>

```
Rect<float> sq = new Rect<float> { Width = 4f, Height = 5f };

var s = sq.Square();
```

Обобщенные типы придуманы для типизации коллекций.

Пространство System.Collections .Generic

Интерфейсы

IEnumerable<T>
IEnumerator<T>
ICollection<T> Размер, перечисление , потокобезопасность

IList<T> Добавление, удаление и индексация
IDictionary<TKey, TValue>
ISet<T>

IComparer<T>

Классы

List<T>

Dictionary<TKey, TValue>

SortedDictionary<TKey, TValue>

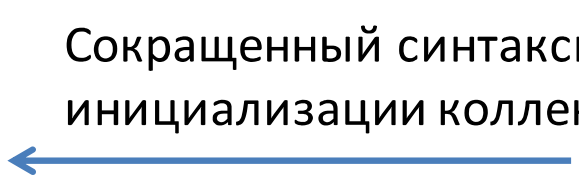
SortedSet<T>
LinkedList<T>
Stack<T>
Queue<T>

Коллекция List<T>

```
class Point
{
    public int X, Y;
}
```

```
List<Point> l = new List<Point>
{
    new Point { X = 3, Y = 3 },
    new Point { X = 2, Y = 4 },
};

l.Sort();

foreach (var p in l)
    Console.WriteLine(p.X);
```

Сокращенный синтаксис инициализации коллекции



Естественный порядок в коллекции

Естественный порядок элементов определяется тем, что элемент реализует интерфейс IComparable или IComparable <T>.

```
class Point: IComparable
{
    public int X, Y;

    public int CompareTo(object obj)
    {
        Point p = (Point)obj;
        return X - p.X;
    }
}
```

```
class Point : IComparable<Point>
{
   public int X, Y;
   public int CompareTo(Point p)
   {
      return X - p.X;
   }
}
```

Навязанный порядок в коллекции

Навязанный порядок задается при помощи объекта с интерфейсом IComparer<T>.

```
class PointComparator : IComparer<Point>
{
    public int Compare(Point a, Point b)
    {
        return a.X - b.X;
    }
 }
```

Сортировка с компаратором:

```
list.Sort(new PointComparator());
```

Ковариантность

```
class B { }
class D : B { }
```

```
class Program
    {
        static void Main()
        {
            B[] b = new D[10];                      // 1.0
            IEnumerable<B> bb = new List<D>();      // 3.5
            IEnumerable<B> bb = new List<D>();      // 4.0
        }
    }
```

В интерфейсе IEnumerable<T> тип T находится только на выходных позициях.

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

Контравариантность

```
class B { }
class D : B { }
```

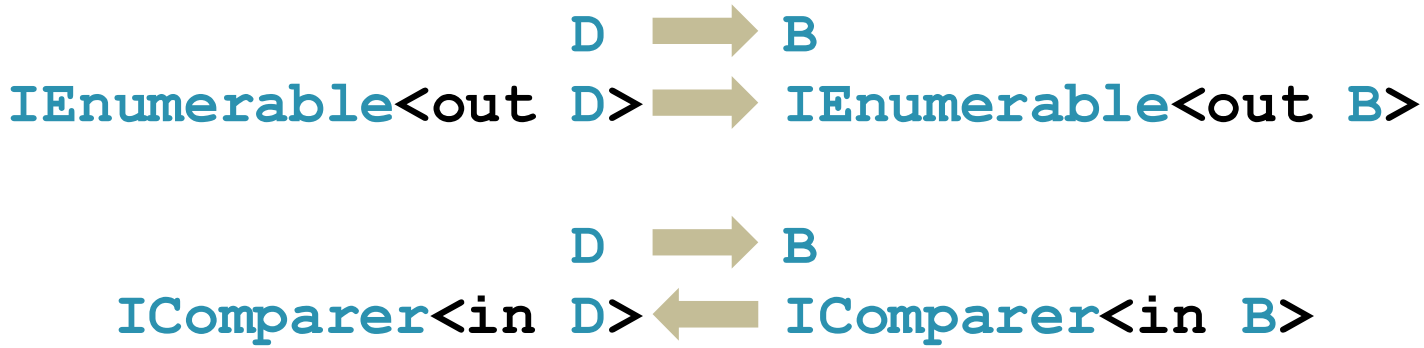
```
class MyComparer<T> : IComparer<T>
{
    public int Compare(T x, T y)
    {
        return ...;
    }
}

class Program
{
    static void Main()
    {
        IComparer<D> cd = new MyComparer<B>();   // 3.5
        IComparer<D> cd = new MyComparer<B>();   // 4.0
    }
}
```

В интерфейсе IComparable<T> тип T находится только на входных позициях (in).

```
public interface IComparer<in T>
```

Почему «ко» и «контра»

Обозначим безопасные преобразования типов стрелками.



Стрелки совпадают – ковариантность.
Стрелки навстречу – контравариантность.

Ко- и контравариантность возможна, только если параметр – ссылочный тип.

Ко- и контравариантность возможна только для обобщенных интерфейсов и обобщенных делегатов.

Самостоятельно

1. Объявить статический метод Freq(), который получает текст и возвращает частотный словарь текста. Слова в тексте разделены произвольным количеством пробелов.

2. Задан список. Оставить в нем только уникальные элементы. Для этого объявить статический метод void Unique<T>(), который получает обобщенный список и убирает из него все вхождения элемента, кроме первого.

3. Объявить обобщенный класс Runner<T>, объект которого можно проходить в цикле foreach. Класс должен иметь конструктор с переменным числом параметров типа T. Эти значения, переданные конструктору, и должны перечисляться в цикле foreach.