

Преговор

- Обобщени типове (Generics);
- Делегати;
- Ламбда изрази;
- Разширяващи методи;
- LINQ;
- Атрибути;
- Библиотеки;
- Отражение (Reflection).

Задача

- Да се направи универсален метод ParseTo.

```
Object ParseTo(Type outType, String text)
```

Тривиална имплементация

```
public Object ParseTo(  
    Type outType,  
    String text)  
{  
    if (outType == typeof(int))  
    {  
        return Int32.Parse(text);  
    }  
    else if (outType == typeof(bool))  
    {  
        return Boolean.Parse(text);  
    }  
    //...  
  
    throw new NotSupportedException();  
}
```

Generics методи

```
public static T ParseTo<T>(
    String text)
{
    if (typeof(T) == typeof(int))
    {
        return (T)(Object)Int32.Parse(text);
    }
    else if (typeof(T) == typeof(bool))
    {
        return (T)(Object)Boolean.Parse(text);
    }
    //...

    throw new NotSupportedException();
}
```

Generics + Reflection + LINQ

```
public static T ParseTo<T>(
    this String text)
{
    var mi = typeof(T)
        .GetMethod()
        .Where(c => c.Name == "Parse")
        .Select(s => new
        {
            MethodInfo = s,
            Parameters = s
                .GetParameters()
        })
        .Where(c =>
            c.Parameters.Count() == 1 &&
            c.Parameters
                .Single().ParameterType == typeof(String))
        .Select(s => s.MethodInfo)
        .SingleOrDefault();
```

```
    if (mi != null)
    {
        return (T)mi.Invoke(
            null,
            new object[] { text });
    }

    throw new NotSupportedException();
}
```

Generics делегати

```
public delegate T Function<T> ();  
public delegate T Function<T1, T>(T1 t1);  
public delegate T Function<T1, T2, T>(T1 t1, T2 t2);
```

```
Function<string, string, string> concat = (s1, s2) => s1 + s2;  
Function<int, int, int> add = (i1, i2) => i1 + i2;
```

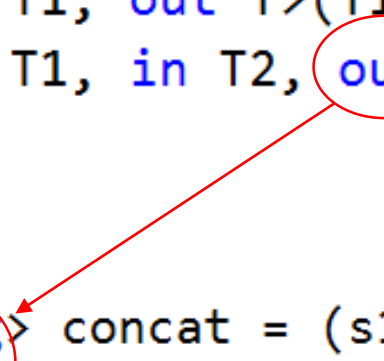
Generics делегати

```
Function<string, string, string> concat = (s1, s2) => s1 + s2;  
Function<string, string, object> b;  
b = concat;
```

Generics делегати

```
public delegate T Function<T> ();  
public delegate T Function<in T1, out T>(T1 t1);  
public delegate T Function<in T1, in T2, out T>(T1 t1, T2 t2);
```

```
Function<string, string, string> concat = (s1, s2) => s1 + s2;  
Function<string, string, object> b;  
b = concat;
```



Задача

- Да се създаде клас (MyList) имплементиращ динамичен масив;
- Да се реализира метода Add на класа.

Generics класове

```
public class MyList<T>
    //where T : Object, new()
{
    private T[] _array = new T[1];
    private int _lastIdx = 0;

    private T Add(T t)
    {
        if (_array.Length >= _lastIdx)
        {
            var newArray = new T[_array.Length * 2];
            Array.Copy(_array, newArray, _lastIdx);
            _array = newArray;
        }

        _array[_lastIdx] = t;
        _lastIdx++;

        return t;
    }
}
```

Generics класове - ограничения

```
public class MyList<T>
    //where T : Object, new()
{
    private T[] _array = new T[1];
    private int _lastIdx = 0;

    private T Add(T t)
    {
        if (_array.Length >= _lastIdx)
        {
            var newArray = new T[_array.Length * 2];
            Array.Copy(_array, newArray, _lastIdx);
            _array = newArray;
        }

        _array[_lastIdx] = t;
        _lastIdx++;

        return t;
    }
}
```

Задача

- Да се реализира универсален ToStringEx метод;
- Метода са показва всички полета на клас декорирани с атрибута ToStringAttribute.

Атрибутът ToString

```
[AttributeUsage(  
    AttributeTargets.Field |  
    AttributeTargets.Property)]  
public class ToStringAttribute : Attribute  
{  
}
```

Extension + Reflection + Attributes + LINQ

```
public static class ToStringExtension
{
    public static String ToStringEx(
        this Object obj)
    {
        return obj
            .GetType()
            .GetMembers()
            .Where(c =>
                c.GetCustomAttribute(
                    typeof(ToStringAttribute)) != null &&
(c is FieldInfo ||
c is PropertyInfo))
            .Select(s =>
                s is FieldInfo ? ((FieldInfo)s).GetValue(obj) :
                s is PropertyInfo ? ((PropertyInfo)s).GetValue(obj) :
                "")
            .Select(s => s.ToString())
            .Aggregate((f, s) => f + "; " + s);
    }
}
```

Задача

- Дадена е колекцията:

```
var ints = new int[] { 2, 4, 1, 10, 3, 7 };
```

- Да се състави LINQ израз връщащ всички четни елементи в масив.

LINQ

```
var ints = new int[] { 2, 4, 1, 10, 3, 7 };  
var even = ints  
    .Where(c => c % 2 == 0)  
    .ToArray();
```


Enumerable.Range()

```
var range1 = Enumerable.Range(0, 4);  
var range2 = new int[] { 0, 1, 2, 3 };
```

Задача

- Дадена е колекцията:

```
var ints = new int[] { 2, 4, 1, 10, 3, 7 };
```

- Да се състави LINQ израз връщащ всички елементи на четни позиции в масив.

LINQ

```
var everysecond = Enumerable.Range(0, ints.Length)
    .Where(c => c % 2 == 0)
    .Select(s => ints[s])
    .ToArray();
```

```
var i = 0;
var everysecond2 = ints
    .Where(c => i++ % 2 == 0)
    .Select(s => ints[i - 1])
    .ToArray();
```

Задача

- Да се определи стойността на x.

```
var ints = new int[] { 2, 4, 1, 10, 3, 7 };  
var x = ints  
    .Where(c => c / 3 > 0)  
    .Select(s2 => s2 + ints  
        .Where(c => c / 3 == 0)  
        .Aggregate((f, s) => f - s))  
    .Sum();
```

Библиотеки

- Библиотеката е асембли (IL code);
- Няма входна точка (метод main);
- Може да има версия (важи за всяко асембли);
- Може да е силно именувана (важи за всяко асембли);
- Може да се инсталира в GAC (важи за всяко асембли).

Библиотеки – кога?

- Когато едни и същи типове се използват многократно в различни проекти. Обемът на кода намалява, промени в логиката изискват промяна само на едно място;
- Когато има относително независими типове.