ШАБЛОНЫ. LINQ.

C# - ЛЕКЦИЯ 11

НА ПРОШЛОМ ЗАНЯТИИ

- Делегаты
- События
- Анонимные функции

СЕГОДНЯ В ПРОГРАММЕ

- Generics
- Extension methods
- LINQ

GENERICS

Generics (универсальные шаблоны)

- Позволяют писать шаблонный код
- Шаблонизация происходит относительно типов данных

Шаблонными могут быть

- Метод
- Класс
- Структура
- Интерфейс
- Делегат

```
static void Swap<T>(ref T a, ref T b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

```
class Program
{
    static void Main(string[] args)
    {
        List<int> seventeens = CreateListOfValues(17, 42);
        List<string> jacks = CreateListOfValues("Jack", 134);
        List<Point> points = CreateListOfValues(new Point(34, 78), 134);
    }

    static List<T> CreateListOfValues<T>(T value, int count)
    {
        var list = new List<T>(count);

        for (int i = 0; i < count; i++)
        {
            list.Add(value);
        }

        return list;
    }
}
```

```
class List<T>
{
    private T[] items;

    public int Count { get; private set; } = 0;

    public MyList()
        : this(4)
    {
    }

    public MyList(int capacity)
    {
        items = new T[capacity];
    }

    public void Add(T item)
    {
        if (Count >= items.Length)
            Array.Resize(ref items, items.Length * 2);

        items[Count] = item;
        Count++;
    }
}
```

GENERICS

Ограничения на шаблонные типы

Ограничения на шаблонные типы

- Позволяют предоставить информацию о шаблонизируемом типе
- Компилятор не даст нарушить ограничения пользователям
- Но разрешит пользоваться информацией разработчику

Ограничения на шаблонные типы

- `where T: struct`
- `where T: class`
- `where T: BaseClassName`
- `where T: InterfaceName`
- `where T: U`
- `where T: new()`

```
void Sort<T>(IList<T> list) where T : IComparable<T>
{
    for (int i = 0; i < list.Count - 1; i++)
        for (int j = i + 1; j < list.Count; j++)
            if (list[i].CompareTo(list[j]) > 0)
            {
                T temp = list[i];
                list[i] = list[j];
                list[j] = temp;
            }
}
```

GENERICS

Covariance, contravariance, invariance

Covariance, contravariance, invariance

- Это дополнительные ограничения на параметры шаблонов
- Можно использовать только в интерфейсах и делегатах

Ковариантность

- Позволяет использовать "более наследованный" тип, чем указано
- Например, можно присвоить экземпляр `IEnumerable<Derived>` переменной типа `IEnumerable<Base>`
- `<out T>` — ковариантный параметр

```
IEnumerable<Derived> d = new List<Derived>();
IEnumerable<Base> b = d;
```

Контрвариантность

- Позволяет использовать "более базовый" тип, чем указано
- Например, можно присвоить экземпляр `Action<Base>` переменной типа `Action<Derived>`
- `<in T>` — ковариантный параметр

```
Action<Base> b = (target) => { Console.WriteLine(target.GetType().Name); };
Action<Derived> d = b;
d(new Derived());
```

Инвариантность

- Не позволяет использовать иной тип, нежели указано
- Например, нельзя присвоить экземпляр `List<Base>` переменной типа `List<Derived>` и наоборот.

EXTENSION METHODS

LINQ

ВОПРОСЫ