

# Разработка .NET приложений

## Лекция 5

### Обобщения (шаблоны)

# Шаблоны, обобщения (Generics)

---

- Тип, метод или интерфейс параметризованный другим типом
- Обобщенный тип
  - Тип (класс, структура), который параметризован другим типом
  - `class Queue<T> { }`
  - `struct Complex<T> { }`
- Обобщенная функция
  - Функция, параметром которой является тип
  - `T Swap<T>(T x, T y) { }`
- Обобщенный делегат
  - Делегат, параметрами которого являются типы
  - `delegate T MyDelegate<T>() where T : new()`
- Обобщенный интерфейс
  - `interface IEnumerable<T> { }`

# Зачем нужны обобщения?

---

## Не типизированные коллекции

- + • Один код для всех
- • Нет контроля типов, накладные расходы

## Коллекции с жёстко заданным типом

- • Каждый раз новый код
- + • Контроль типов, нет накладных расходов

## Обобщенные коллекции

- + • Один код для всех
- + • Контроль типов, нет накладных расходов

# Синтаксис обобщенного типа

```
[Модификаторы] class ИмяКласса<ОбозначенияТипов>  
[Ограничения типов] {...}
```

ОбозначенияТипов := Обозначение1 [,ОбозначениеN]

Ограничения типов := **where** ОбозначениеТипа1 : ограничения  
[, ОбозначениеТипаN : ограниченияN]

## Пример:

```
public class Stack<T>  
    where T : IDisposable, new()  
{  
    T[] arr;  
    T top = default(T);  
    T example = new T();  
    Stack(int size) { arr = new T[size]; }  
    void Push(T item) {...}  
    T Pop() {return arr[i]; }  
    void Dispose() {top.Dispose();}  
}
```

## Создание объекта обобщенного типа:

Необходимо указать все типы-параметры

```
Stack<int> myIntStack = new Stack<int>();
```

```
Dictionary<int, string> myDict = new Dictionary<int, string> ();
```

# Примеры

---

- Типов-параметров может быть много
- Обозначения типов –любые (допустимые) имена (Часто обозначают как TНазвание)
- Обозначения типов могут использоваться в коде как обычный тип
- Для задания значения по умолчанию используется синтаксис
- Тип Переменная = `default(Тип );`

```
class MyDictionary<TKey, TValue>  
    where TKey : Complex, IComparable  
    where TValue : class  
{  
}
```

```
struct MyGenericStruct<T>  
{  
}
```

```
class MyGenericClass<A,B,C,B,E,F>  
{  
    C c = default(C);  
}
```

# Демонстрации

---

Обобщенный список

# Синтаксис обобщенной функции

*[Модификаторы]*

*ВозвращаемыйТип* ИмяМетода **<ОбозначенияТипов>**(параметры)

*[Ограничения типов]* { }

- Примеры:

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

```
IEnumerable<T> Concat<T>( IEnumerable<T> first, IEnumerable<T> second ) { ... }
```

```
public T Copy<T>(T a, T b)
    where T : new()
{ T result = new T(); .....}
```

- Использование:

- При вызове необходимо задать все типы параметры, если компилятор не может определить их из контекста  
`Swap<Complex>(ref comp1, ref comp2);`  
`int I = Do<int, long>(intVariable, longVariable, myIntComplexVariable);`

# Обобщенный интерфейс

[Модификаторы] **interface** ИмяИнтерфейса<ОбозначенияТипов>  
[Ограничения типов] {...}

- **Пример:**

```
public interface IEntityProvider<TEntity>
{
    TEntity[] Load();
    void Save(TEntity[] entities);
    TEntity GetById(int id);
    void DeleteAll();
}
```

- **Использование:**

- При реализации, либо класс должен быть обобщенным, либо должен реализовывать конкретный тип обобщенного интерфейса

```
class ProductProvider : IEntityProvider<Product> { }
class DbEntityProvider<TEntity> : IEntityProvider<TEntity> { }
class FileEntityProvider<T> : IEntityProvider<T> { }
```

# Ограничения на параметры

---

- ◎ Типы-параметры:

- если что-то явно не оговорено, то этого НЕТ!
- По умолчанию доступны только методы **object**

- ◎ Если нужны дополнительные методы

- Необходимо использовать ограничения

# Типы ограничений

---

- Интерфейсы
  - **where** T : ICloneable, IDisposable
- Базовый класс
  - **where** T : MyBaseClass
  - Исключения: Нельзя указывать Object, Array, и ValueType
- Ссылочные типы / Типы значения
  - **where** T : struct
  - **where** T : class
  - Нельзя одновременно указать и class, и struct
- Наличие конструктора по умолчанию
  - ограничение должно быть последним в списке
  - **where** T : new()
- Требования использования типа не допускающего значения **null** (C#8+)
  - **where** T : notnull
  - Могут быть типы значения и ссылочные типы (при включённом контексте явного указания nullable ссылочных типов)
- Требования использования **unmanaged** типа
  - **where** T : unmanaged // всегда struct

# Ограничения ограничений

---

- Нельзя требовать наличие операции
  - Дублируйте обычными методами
- Нельзя требовать конструктора с параметрами
- Если указано несколько ограничений
  - Все должны выполняться одновременно
  - Если нужна альтернатива – используйте базовые интерфейсы
- Ограничение базового класса должно быть указано перед любыми другими ограничениями
- Нельзя использовать в качестве ограничения базовые классы `Object`, `Array` и `ValueType`

# Демонстрации

---

Сортировка пузырьком массива произвольного типа