Обобщения

Мухортова Н.Н.

Цель

Освоить инструмент ООП - обобщения. Научиться проектировать классы с использованием обобщений.

Generics

Кроме обычных типов фреймворк .NET также поддерживает обобщенные типы (generics), а также создание обобщенных методов.

```
class Account<T>
{
    public T Id { get; set; }
    public int Sum { get; set; }
}
```

https://metanit.com/sharp/tutorial

Обращение к классу

```
Account<int> account1 = new Account<int> { Sum = 5000 };
Account<string> account2 = new Account<string> { Sum = 4000 };
account1.Id = 2;        // упаковка не нужна
account2.Id = "4356";
int id1 = account1.Id;  // распаковка не нужна
string id2 = account2.Id;
Console.WriteLine(id1);
Console.WriteLine(id2);
```

Значения по умолчанию

Иногда возникает необходимость присвоить переменным универсальных параметров некоторое начальное значение, в том числе и null. Но напрямую мы его присвоить не можем.

В этом случае нам надо использовать оператор default(T). Он присваивает ссылочным типам в качестве значения null, а типам значений - значение 0:

```
class Account<T>
{
    T id = default(T);
}
```

Статические поля обобщенных классов

При типизации обобщенного класса определенным типом будет создаваться свой набор статических членов. Например, в классе Account определено следующее статическое поле:

```
class Account<T>
{
    public static T session;
    public T Id { get; set; }
    public int Sum { get; set; }
}
```

В итоге для Account<string> и для Account<int> будет создана своя переменная session.

Использование нескольких универсальных параметров

```
class Transaction<U, V>
{
    public U FromAccount { get; set; }  // с какого счета перевод
    public U ToAccount { get; set; }    // на какой счет перевод
    public V Code { get; set; }         // код операции
    public int Sum { get; set; }        // сумма перевода
}
```

Универсальные классы

Инкапсулируют операции, которые не относятся к конкретному типу данных.

Чаще всего используются для работы с коллекциями, такими как связанные списки, хэш-таблицы, стеки, очереди, деревья и т. д. Такие операции, как добавление и удаление элементов коллекции, по существу выполняются одинаково, независимо от типа хранимых данных.

Наследование

Универсальные классы могут наследоваться от конкретных, а также закрытых или открытых сконструированных базовых классов:

```
class BaseNode { }

class BaseNodeGeneric<T> { }


class NodeConcrete<T> : BaseNode { }

class NodeClosed<T> : BaseNodeGeneric<int> { }

class NodeOpen<T> : BaseNodeGeneric<T> { }
```

Наследование

Классы, не являющиеся универсальными, то есть конкретные классы, могут наследоваться от закрытых сконструированных базовых классов.

```
class Node1 : BaseNodeGeneric<int> { }
```

Наследование

```
class BaseNodeMultiple<T, U> { }

class Node4<T> : BaseNodeMultiple<T, int> { }
class Node5<T, U> : BaseNodeMultiple<T, U> { }
```

Ограничения

Универсальные типы могут использовать несколько параметров типа и ограничений, как показано ниже:

```
class SuperKeyType<K, V, U>
    where U : System.IComparable<U>
    where V : new()
{}
```

Параметры

Открытые и закрытые сконструированные типы можно использовать в качестве параметров метода:

```
void Swap<T>(List<T> list1, List<T> list2)
{    ….
 }
void Swap(List<int> list1, List<int> list2)
{    ....
 }
```

Универсальные интерфейсы

В качестве ограничений для одного типа можно задать несколько интерфейсов, как показано ниже:

```
class Stack<T> where T : System.IComparable<T>, IEnumerable<T>
{
}
```

Универсальные интерфейсы

Интерфейс может определять несколько параметров типа, как показано ниже:

```
interface IDictionary<K, V>
{
}
```

Универсальные интерфейсы

Правила наследования, действующие в отношении классов, также применяются к интерфейсам:

```
interface IMonth<T> { }

interface IJanuary     : IMonth<int> { }  //No error

interface IFebruary<T> : IMonth<int> { }  //No error

interface IMarch<T>    : IMonth<T> { }    //No error
```

Универсальные интерфейсы могут наследоваться от неуниверсальных интерфейсов, если универсальный интерфейс является контравариантным (то есть использует в качестве возвращаемого значения только свой параметр типа)

Универсальные интерфейсы

Универсальные классы могут реализовывать универсальные интерфейсы или закрытые сконструированные интерфейсы при условии, что в списке параметров класса заданы все аргументы, требуемые интерфейсом, как показано ниже:

```
interface IBaseInterface1<T> { }

interface IBaseInterface2<T, U> { }


class SampleClass1<T> : IBaseInterface1<T> { }

class SampleClass2<T> : IBaseInterface2<T, string> { }
```

Универсальные (Обобщенные) методы

```
public static void Swap<T> (ref T x, ref T y)
    {
        T temp = x;
        x = y;
        y = temp;
    }
Swap<int>(ref x, ref y); // или так Swap(ref x, ref y);
Swap<string>(ref s1, ref s2); // или так Swap(ref s1, ref s2);
```

Универсальные (Обобщенные) методы

Для большей гибкости можно вызывать метод универсального класса с использованием аргументов типа, отличающихся от предоставленных при создании экземпляра класса. В этом случае следует предоставить другой идентификатор для параметра типа метода

```
class GenericList2<T>
{
    //No warning
    void SampleMethod<U>() { }
}
```

Перегрузка

Универсальные методы могут быть перегружены в нескольких параметрах типа. Например, все представленные ниже методы могут располагаться в одном классе:

```
void DoWork() { }

void DoWork<T>() { }

void DoWork<T, U>() { }
```

Выводы

Кроме обычных типов фреймворк .NET также поддерживает обобщенные типы (generics), а также создание обобщенных методов.

Угловые скобки в описании class Person<T> указывают, что класс является обобщенным, а тип T, заключенный в угловые скобки, будет использоваться этим классом. Причем на этапе написания кода неизвестно, что это будет за тип, это может быть любой тип. Поэтому параметр T в угловых скобках еще называется универсальным параметром, так как вместо него можно подставить любой тип.