ООП в C#

(универсальные методы, делегаты)

Андрей Голяков

Универсальные методы (generic methods)

Универсальным (обобщённым) называется метод, объявленный с использованием параметризованного типа данных.

Например, у нас есть статический класс, метод которого выполняет определённую логику. Пусть это будет метод Swap, меняющий 2 переменные местами.

```
public static class Swapper
{
    public static void Swap(int a, int b)
    {
        int temp = a;
        a = b;
        b = temp;
    }
}
```

Универсальные методы (generic methods)

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

class Program
{
      static void Main(string[] args)
      {
            int a = -1;
            int b = 100;
            Console.WriteLine($"a = {a}, b = {b}");
            Swapper.Swap<int>(ref a, ref b);
            Console.WriteLine($"a = {a}, b = {b}");
      }
}
```

Универсальные классы (generic classes)

Универсальные классы инкапсулируют операции, которые не относятся к конкретному типу данных.

Например, наш класс Swapper можно сделать универсальным, перенеся параметр T уже на уровень класса

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

Правила именования

Основные правила:

- Присваивайте параметрам универсального типа описательные имена, кроме случаев, когда достаточно одной буквы и описательное имя не имеет практической ценности.
- Используйте имя параметра типа T для типов, содержащих только однобуквенный параметр типа.
- Используйте префикс "T" для описательных имен параметров типа (TSession)
- Указывайте ограничения, связанные с параметром типа, в его имени. Например, параметр с ограничением ISession может называться TSession.

Полный список с примерами можно найти в документации от Майкрософ

Самостоятельная работа

Реализовать класс Account

- Два свойства:
  - int Id { get, private set },
  - string Name { get, private set },
- Конструктор с параметрами для их заполнения
- Публичный метод void WriteProperties(), выводящий параметры в консоль.

Затем сделать класс обобщённым, обобщяя его по типу свойства Id.

В основном потоке программы создать несколько экземпляров класса Account с различными типами данных для Id: int, string, Guid. Для всех вызвать метод WriteProperties.

Ограничения параметров типа

Ограничения сообщают компилятору о характеристиках, которые должен иметь аргумент типа.

Без ограничений аргумент типа может быть любым типом,

Дополнительные сведения см. в статье Зачем использовать ограничения.

Если в клиентском коде для создания экземпляра класса используется недопустимый тип, возникает ошибка времени компиляции.

Ограничения задаются с помощью контекстного ключевого слова where.

Ограничения параметров типа

Основные типы ограничений:

where T : struct Аргумент типа должен быть типом значения. Допускается задание любого типа значения, кроме Nullable<T>.

where T : class Аргумент типа должен быть ссылочным типом. Это ограничение также применяется к любому типу класса, интерфейса, делегата или массива.

where T : new() Аргумент типа должен иметь общий конструктор без параметров. При одновременном использовании нескольких ограничений последним должно указываться ограничение new().

Ограничения параметров типа

Основные типы ограничений:

where T : <имя базового класса> Аргумент типа должен иметь базовый класс или производный от него класс.

where T : <имя интерфейса> Аргумент типа должен являться заданным интерфейсом или реализовывать его. Можно указать несколько ограничений интерфейса. Заданный в ограничении интерфейс также может быть универсальным.

Синтаксис:

```
public static class Swapper<T> where T: struct
```

Делегаты

Делегат — это тип, который представляет ссылку на метод(ы) с определенным списком параметров и типом возвращаемого значения.

При создании экземпляра делегата этот экземпляр можно связать с любым методом с совместимой сигнатурой и типом возвращаемого значения. Метод можно вызвать (активировать) с помощью экземпляра делегата.

Делегаты используются для передачи методов в качестве аргументов к другим методам.

Делегат можно связать с именованным методом. При создании экземпляра делегата с использованием именованного метода этот метод передается в качестве параметра в конструктор делегата.

Делегаты

По факту же при компиляции кода в CIL — компилятор превращает каждый такой тип-делегат в одноименный тип-класс и все экземпляры данного типа-делегата по факту являются экземплярами соответствующих типов-классов.

Объявляя новый тип-делегат мы сразу через синтаксис его объявления жестко определяем сигнатуру допустимых методов, которыми могут быть инициализированы экземпляры такого делегата.

Экземпляр же такого делегата стоит понимать как ссылку на конкретный метод или список методов, который куда то будет передан и скорее всего выполнен уже на той стороне. Причем не сможет передать с методом значение аргументов с которыми он будет выполнен, или поменять его сигнатуру. Но он сможет определить логику работы метода — его тело.

Делегаты с именованными методами

```
public class SimpleCalculator
{
     public int Sum(int x, int y)
     {
          return x + y;
     }

     public int Multiply(int x, int y)
     {
          return x * y;
     }
}
```

```
delegate int PerformCalculation(
     int x, int y);

static void Main(string[] args)
{
     var calc = new SimpleCalculator();
     PerformCalculation performCalculation;
     int result;

     performCalculation = calc.Sum;
     result = performCalculation(10, 5);
     Console.WriteLine(result);

     performCalculation = calc.Multiply;
     result = performCalculation(10, 5);
     Console.WriteLine(result);
}
```

Делегаты: ссылки на несколько методов

Можно создать цепочку вызовов, когда делегат хранит ссылки на более чем один метод. Для этого нужно вызвать статический метод Delegate.Combine или оператор +=

```
DoCalculation doCalculation1 = SimpleCalculator.Sum;
DoCalculation doCalculation2 = SimpleCalculator.Multiply;

// operator +=
doCalculation1 += doCalculation2;

// static method Delegate.Combine
doCalculation1 = (DoCalculation)Delegate.Combine(
    doCalculation1, doCalculation2);
```

Делегаты: ссылки на несколько методов

Можно создать цепочку вызовов, когда делегат хранит ссылки на более чем один метод. Для этого нужно вызвать статический метод Delegate.Combine или оператор +=

При выполнении такого делегата все его методы будут выполнены по очереди. Если сигнатура делегата предполагает получение параметров то параметры будут для всех методов иметь одно значение. Если есть возвращаемое значение, то мы можем получить лишь значение последнего в списке метода.

```
DoCalculation doCalculation1 = SimpleCalculator.Sum;
DoCalculation doCalculation2 = SimpleCalculator.Multiply;

// operator +=
doCalculation1 += doCalculation2;

// static method Delegate.Combine
doCalculation1 = (DoCalculation)Delegate.Combine(
     doCalculation1, doCalculation2);

var multipleResult = doCalculation1(10, 5);
Console.WriteLine($"{nameof(multipleResult)} = {multipleResult}");
```

Делегаты: ссылки на несколько методов

Метод Delegate.Remove() или переопределённый оператор -= в свою очередь производит поиск в списке делегатов по значению объекта-владельца и методу, и в случае нахождения удаляет первый совпавший.

```
// operator -=
doCalculation1 -= doCalculation2;

// method Remove()
doCalculation1 = (DoCalculation)Delegate.Remove(
     doCalculation1, doCalculation2);

multipleResult = doCalculation1(10, 5);
Console.WriteLine($"{nameof(multipleResult)} = {multipleResult}");
```

Делегаты библиотеки классов .NET

Стоит сказать что делегаты могут быть обобщенными (Generic), что является более правильным подходом к созданию отдельных делегатов для разных типов.

Также стоит упомянуть что библиотека .NET Framework Class Library уже содержит наиболее популярные типы делегатов (обобщенные и нет). Например:

- Делегат Action<T> представляет собой метод без возвращаемого значения но с аргументом,
- Делегат Func<T, TResult> и с возвращаемым значением и аргументом.

Делегаты

Полезные ссылки:

- Habr: Зачем нужны делегаты в C#?
- Habr: Делегаты и Лямбда выражения в C#

Самостоятельная работа

Дан класс:

```
public sealed class Circle
{
	private double _radius;

	public Circle(double radius)
	{
		_radius = radius;
	}

	public double Calculate(Func<double, double> operation)
	{
		return operation(_radius);
	}
}
```

Не изменяя данного класса

- Реализовать возможность вычисления периметра и площади окружности.
- Создать экземпляр класса с радиусом = 1.5
- Рассчитать периметр и площадь пользуясь методом Calculate() у созданного экземпляра класса.
- Вывести на экран результат вычислений в формате:

For the circle with radius 1.5
	Perimeter is 9.42477796076938
	Square is 7.06858347057703

Домашнее задание

Реализовать класс LogWriterFactory, который бы сам реализовывал паттерн синглтон создавал и выдавал экземпляры классов с позапрошлого (не синглтоны!) домашнего задания:

- ConsoleLogWriter,
- FileLogWriter,
- MultipleLogWriter.

Класс должен иметь один метод

```
public ILogWriter GetLogWriter<T>(object parameters) where T : ILogWriter
```

и возвращать любой из трёх возможных классов выше (сами классы не должны меняться).

В основном потоке программы:

- Создать по одному экземпляру класса FileLogWriter и ConsoleLogWriter.
- Затем создать экземпляр класса MultipleLogWriter, который бы принял в конструктор созданные выше экземпляры FileLogWriter и ConsoleLogWriter.
- Сделать по одной записи логов каждого типа, чтобы убедиться, что они одновременно пишутся и в консоль и в файл.

Спасибо за внимание.