Темы урока

Универсальные методы

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.

Пример Swapper

Описываем проблему - метод реализует алгоритм не зависящий от типа переменных. Однако, из-за строгой типизации мы должны описать его для каждого типа данных отдельно.

Продемонстрировать на примере **L15_C01_generic_methods** как разрастётся код, если дублировать его для всех типов данных.

Как вариант показать, что **можно использовать object**, но **указать на минусы** упаковки/распаковки.

Написать метод Swap<T> (пример можно посмотреть в **L15_C02_generic_methods_final**)

Потом реализовать вызов для **int**, для **float**, для **string**, для **int[]**.

Конечный результат можно посмотреть в **L15_C02_generic_methods_final.**

Универсальные классы

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

Решение: **L15_C03_generic_classes**.

Самостоятельная работа

Написать класс Account (задание на слайде).
Решение: **L15_C04_generic_classes_SW**.

Ограничения

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

Без ограничений аргумент типа может быть любым типом,

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

Ограничения задаются с помощью контекстного ключевого слова **where**.

Зачем использовать ограничения?

Ограничивая параметр типа, вы расширяете выбор разрешенных операций и вызовов метода до поддерживаемых ограничивающим типом и всеми типами в его иерархии наследования.

Если при разработке универсальных классов или методов планируется выполнение любых других операций с универсальными элементами, помимо простого присвоения или вызова методов, не поддерживаемых System.Object, вам необходимо применить ограничения к параметру типа.

Например, на основе ограничения базового класса компилятор определяет, что в качестве аргументов типа будут использоваться только объекты указанного типа или производных от него. Имея такую гарантию, компилятор может вызывать методы указанного типа в универсальном классе.

Делегаты

Делегат — это тип, который представляет ссылки на методы 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

Ссылки на несколько методов

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

Пример на слайде и **L15_C06_delegate_demo**.

Множественные методы в делегатах

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.

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.

Пример на слайдах и **L15_C07_delegate_multiple_demo**.

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.

Обратить внимание, что **есть много версий для разного количества параметров**
как у Funk, так и Action.

Пример

Разбор примера (приведение кода в **L15_C08_delegate_fcl_action_func_demo** к коду в **L15_C09_delegate_fcl_action_func_final**).

Дан код решения задачи сложения чисел в массиве и вывода их суммы на экран.

По шагам

1. Выделить методы и оформить их в виде:

   a. static int **Sum**(int[] numbers)
   b. static void **WriteToConsole**(string message)
2. Добавить в Program ещё один метод:
   static void **DoCalcAndWriteOutput**(
      int[] numbers,
      Func<int[], int> doCalc,
      Action<string> writeOutput)
3. В основном потоке программы добавить использовать метод
   DoCalcAndWriteOutput(numbers, Sum, WriteToConsole);
4. Продемонстрировать простоту расширения функционала.
   Добавить в Program ещё один метод
      static int **Avg**(int[] numbers) для подсчёта среднего значения.
5. В основном потоке программы добавить использовать метод
   DoCalcAndWriteOutput(numbers, Avg, WriteToConsole);

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 is9.42477796076938
      Square is         7.06858347057703
```

Решение: **L15_C10_delegate_SW**.

Домашнее задание

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

- **ConsoleLogWriter**,
- **FileLogWriter**,
- **MultipleLogWriter**.

Класс должен быть 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.