

№ 8 Обобщения

Задание

1. Создайте **обобщенный интерфейс** с операциями добавить, удалить, просмотреть.
2. Возьмите за основу лабораторную № 4 «Перегрузка операций» и сделайте из нее **обобщенный тип (класс) `CollectionType<T>`**, в который вложите обобщённую коллекцию. Наследуйте в обобщенном классе интерфейс из п.1. Реализуйте необходимые методы. Добавьте **обработку исключений** с `finally`. Наложите какое-либо **ограничение** на обобщение.
3. Проверьте использование обобщения для стандартных типов данных (в качестве стандартных типов использовать целые, вещественные и т.д.). Определите пользовательский класс, который будет использоваться в качестве параметра обобщения. Для пользовательского типа взять класс из лабораторной №5 «Наследование».
4. Добавьте методы сохранения объектов обобщённого типа `CollectionType<T>` в файл и чтения из него (на выбор: текстовый | xml | json).

Вопросы

1. Что такое обобщение (generic)?
2. Пусть дан фрагмент листинга. В какой строчке содержится ошибка?

```
class Gen<T,G> //1
{
    G ob;//2
    T bo;//3
    public Gen(G o) { ob = o; } //4
    public T GetOb() { return bo; } //5
}
```

3. Как можно наложить определенное ограничение на параметр?
4. Как можно наложить несколько ограничений на параметр?
5. Перечислите все существующие ограничения на типы данных обобщения?
6. Какое ограничение на тип задано в следующем фрагменте листинга?

```
class A { }
class B : A { }
class C { }
class Test<T> where T : A { }
```

7. Какое ограничение на тип задано в следующем фрагменте листинга?

```
interface A { }  
class Test<T> where T : class { }
```

8. Какое ограничение на тип задано в следующем фрагменте листинга?

```
interface A { }  
class Test<T> where T : struct { }
```

9. Приведите примеры, когда обобщенный класс может действовать как базовый или производный класс.
10. В каких случаях в обобщениях может использоваться оператор default?
11. Поясните как использовать статические переменные в обобщенных классах.
12. Приведите пример обобщенного интерфейса.
13. В чем отличие обобщенных классов от обобщенных структур?
14. Какие классы для работы с файлами вы знаете? Приведите пример

Краткие теоретические сведения

Обобщения

Термин *обобщение* означает параметризованный тип. Особая роль параметризованных типов состоит в том, что они позволяют создавать классы, структуры, интерфейсы, методы и делегаты, в которых обрабатываемые данные указываются в виде параметра. Схожие с обобщениями черты имеют шаблоны C++. Однако между шаблонами C++ и обобщениями .NET есть большая разница. В C++ при создании экземпляра шаблона с конкретным типом необходим исходный код шаблонов. В отличие от шаблонов C++, обобщения являются не только конструкцией языка C#, но также определены для CLR. Это позволяет создавать экземпляры шаблонов с определенным типом-параметром на языке Visual Basic, даже если обобщенный класс определен на C#.

Общая форма объявления обобщенного класса:

```
class имя_класса<список_параметров_типа> { // ...
```

А вот как выглядит синтаксис объявления ссылки на обобщенный класс:

```
имя_класса<список_аргументов_типа> имя_переменной =  
new имя_класса<список_параметров_типа> (список_аргументов_конструктора);
```

Рассмотрим пример использования нескольких обобщенных классов:

```
namespace ConsoleApplication1
{
    // Создадим обобщенный класс имеющий параметр типа T
    class MyObj<T>
    {
        T obj;

        public MyObj(T obj)
        {
            this.obj = obj;
        }

        public void objectType()
        {
            Console.WriteLine("Тип объекта: " + typeof(T));
        }
    }

    // Обобщенный класс с несколькими параметрами
    class MyObjects<T, V, E>
    {
        T obj1;
        V obj2;
        E obj3;

        public MyObjects(T obj1, V obj2, E obj3)
        {
            this.obj1 = obj1;
            this.obj2 = obj2;
            this.obj3 = obj3;
        }

        public void objectsType()
        {
            Console.WriteLine("\nТип объекта 1: " + typeof(T) +
                "\nТип объекта 2: " + typeof(V) +
                "\nТип объекта 3: " + typeof(E));
        }
    }

    class Program
    {
        static void Main()
        {
            // Создадим экземпляр обобщенного класса типа int
            MyObj<int> obj1 = new MyObj<int>(25);
            obj1.objectType();

            MyObjects<string, byte, decimal> obj2 = new MyObjects<string,
byte, decimal>("Alex", 26, 12.333m);
            obj2.objectsType();
            Console.ReadLine();
        }
    }
}
```

```
}  
}
```

Когда для класса `MyObj` указывается аргумент типа, например, `int` или `string`, то создается так называемый в C# закрыто сконструированный тип. В частности, `MyObj<int>` является закрыто сконструированным типом. А конструкция, подобная `MyObj<T>`, называется в C# открыто сконструированным типом, поскольку в ней указывается параметр типа `T`, но не такой конкретный тип, как `int`.

Ограниченные типы

Допустим, что требуется создать метод, оперирующий содержимым потока, включая объекты типа `FileStream` или `MemoryStream`. На первый взгляд, такая ситуация идеально подходит для применения обобщений, но при этом нужно каким-то образом гарантировать, что в качестве аргументов типа будут использованы только типы потоков, но не `int` или любой другой тип. Кроме того, необходимо уведомить компилятор о том, что методы, определяемые в классе потока, будут доступны для применения. Так, в обобщенном коде должно быть каким-то образом известно, что в нем может быть вызван метод `Read()`.

Для выхода из подобных ситуаций в C# предусмотрены ограниченные типы. Указывая параметр типа, можно наложить определенное ограничение на этот параметр. Это делается с помощью оператора `where` при указании параметра типа:

```
class имя_класса<параметр_типа> where параметр_типа : ограничения { // ...
```

где ограничения указываются списком через запятую.

Связь между параметрами типа с помощью ограничений

Существует разновидность ограничения на базовый класс, позволяющая установить связь между двумя параметрами типа. В качестве примера рассмотрим следующее объявление обобщенного класса:

```
class MyObj<T, V> where V : T { ...
```

В этом объявлении оператор `where` уведомляет компилятор о том, что аргумент типа, привязанный к параметру типа `V`, должен быть таким же, как и аргумент типа, привязанный к параметру типа `T`, или же наследовать от него. Если подобная связь отсутствует при объявлении объекта типа `MyObj`, то во время компиляции возникнет ошибка. Такое ограничение на параметр типа называется *неприкрытым ограничением типа*.

С параметром типа может быть связано несколько ограничений. В этом случае ограничения указываются списком через запятую. В этом списке

первым должно быть указано ограничение `class` либо `struct`, если оно присутствует, или же ограничение на базовый класс, если оно накладывается. Указывать ограничения `class` или `struct` одновременно с ограничением на базовый класс не разрешается. Далее по списку должно следовать ограничение на интерфейс, а последним по порядку — ограничение `new()`.

Например, следующее объявление считается вполне допустимым:

```
class MyObj<T> where T : MyClass, IMyInterface, new() {  
    // ...
```

В данном случае параметр типа `T` должен быть заменен аргументом типа, наследующим от класса `MyClass`, реализующим интерфейс `IMyInterface` и использующим конструктор без параметра.

Если же в обобщении используются два или более параметра типа, то ограничения на каждый из них накладываются с помощью отдельного оператора `where`.

Обзор коллекций

В **C# коллекция** представляет собой совокупность объектов. В среде .NET Framework имеется немало интерфейсов и классов, в которых определяются и реализуются различные типы коллекций.

Главное преимущество коллекций заключается в том, что они стандартизируют обработку групп объектов в программе. Все коллекции разработаны на основе набора четко определенных интерфейсов. Некоторые встроенные реализации таких интерфейсов, в том числе `ArrayList`, `Hashtable`, `Stack` и `Queue`, могут применяться в исходном виде и без каких-либо изменений. Имеется также возможность реализовать собственную коллекцию, хотя потребность в этом возникает редко.

В среде .NET Framework поддерживаются пять типов коллекций: необобщенные, специальные, с поразрядной организацией, обобщенные и параллельные.

Необобщенные коллекции

Реализуют ряд основных структур данных, включая динамический массив, стек, очередь, а также словари, в которых можно хранить пары "ключ-значение". В отношении необобщенных коллекций важно иметь в виду следующее: они оперируют данными типа `object`. Таким образом, необобщенные коллекции могут служить для хранения данных любого типа, причем в одной коллекции допускается наличие разнотипных данных. Очевидно, что такие коллекции не типизированы, поскольку в них хранятся ссылки на данные типа `object`. Классы и интерфейсы необобщенных коллекций находятся в пространстве имен **System.Collections**.

Специальные коллекции

Опереируют данными конкретного типа или же делают это каким-то особым образом. Например, имеются специальные коллекции для символьных

строк, а также специальные коллекции, в которых используется однонаправленный список. Специальные коллекции объявляются в пространстве имен **System.Collections.Specialized**.

Поразрядная коллекция

В прикладном интерфейсе Collections API определена одна коллекция с поразрядной организацией — это BitArray. Коллекция типа BitArray поддерживает поразрядные операции, т.е. операции над отдельными двоичными разрядами, например И, ИЛИ, исключающее ИЛИ, а следовательно, она существенно отличается своими возможностями от остальных типов коллекций. Коллекция типа BitArray объявляется в пространстве имен System.Collections.

Обобщенные коллекции

Обеспечивают обобщенную реализацию нескольких стандартных структур данных, включая связанные списки, стеки, очереди и словари. Такие коллекции являются типизированными в силу их обобщенного характера. Это означает, что в обобщенной коллекции могут храниться только такие элементы данных, которые совместимы по типу с данной коллекцией. Благодаря этому исключается случайное несовпадение типов. Обобщенные коллекции объявляются в пространстве имен **System.Collections.Generic**.

Параллельные коллекции

Поддерживают многопоточный доступ к коллекции. Это обобщенные коллекции, определенные в пространстве имен **System.Collections.Concurrent**.

В пространстве имен System.Collections.ObjectModel находится также ряд классов, поддерживающих создание пользователями собственных обобщенных коллекций.

Основополагающим для всех коллекций является понятие *перечислителя*, который поддерживается в необобщенных интерфейсах IEnumerator и IEnumerable, а также в обобщенных интерфейсах IEnumerator<T> и IEnumerable<T>. Перечислитель обеспечивает стандартный способ поочередного доступа к элементам коллекции. Следовательно, он перечисляет содержимое коллекции. В каждой коллекции должна быть реализована обобщенная или необобщенная форма интерфейса IEnumerable, поэтому элементы любого класса коллекции должны быть доступны посредством методов, определенных в интерфейсе IEnumerator или IEnumerator<T>. Это означает, что, внося минимальные изменения в код циклического обращения к коллекции одного типа, его можно использовать для аналогичного обращения к коллекции другого типа. Любопытно, что для поочередного обращения к содержимому коллекции в цикле foreach используется перечислитель.

С перечислителем непосредственно связано другое средство, называемое *итератором*. Это средство упрощает процесс создания классов коллекций, например специальных, поочередное обращение к которым организуется в цикле foreach.

Классы коллекций по своей сути подобны классам стандартной библиотеки шаблонов (Standard Template Library — STL), определенной в C++. То, что в программировании на C++ называется контейнером, в программировании на C# называется коллекцией.