Теория и практика программирования

Шпилёв Пётр Валерьевич

Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра статистического моделирования

Лекция 11

Санкт-Петербург
2015 г.

e-mail: pitshp@hotmail.com

Общая форма ограничения ссылочного типа

where T : class

В этой форме с оператором where ключевое слово class указывает на то, что аргумент T должен быть ссылочного типа.

Общая форма ограничения ссылочного типа

where T : class

В этой форме с оператором where ключевое слово class указывает на то, что аргумент T должен быть ссылочного типа.

Общая форма ограничения типа значения

where T : struct

В этой форме ключевое слово struct указывает на то, что аргумент T должен быть типа значения.

Общая форма ограничения ссылочного типа

where T : class

В этой форме с оператором where ключевое слово class указывает на то, что аргумент T должен быть ссылочного типа.

Общая форма ограничения типа значения

where T : struct

В этой форме ключевое слово struct указывает на то, что аргумент T должен быть типа значения.

Упражнение 11.1

Продемонстрировать наложение ограничения ссылочного типа.

Общая форма ограничения ссылочного типа

where T : class

В этой форме с оператором where ключевое слово class указывает на то, что аргумент T должен быть ссылочного типа.

Общая форма ограничения типа значения

where T : struct

В этой форме ключевое слово struct указывает на то, что аргумент T должен быть типа значения.

Упражнение 11.1

Продемонстрировать наложение ограничения ссылочного типа.

Упражнение 11.2

Продемонстрировать наложение ограничения типа значения.

Связь между двумя параметрами типа

```
class Gen<T, V> where V : T
```

Связь между двумя параметрами типа

```
class Gen<T, V> where V : T
```

Упражнение 11.3

Реализовать пример использования связи между двумя параметрами типа.

Связь между двумя параметрами типа

```
class Gen<T, V> where V : T
```

Упражнение 11.3

Реализовать пример использования связи между двумя параметрами типа.

Применение нескольких ограничений

```
class Gen<T> where T : Ограничение_1, Ограничение_2, Ограничение_3
```

Связь между двумя параметрами типа

```
class Gen<T, V> where V : T
```

Упражнение 11.3

Реализовать пример использования связи между двумя параметрами типа.

Применение нескольких ограничений

```
class Gen<T> where T : Ограничение_1, Ограничение_2, Ограничение_3
```

Пример:

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

Пример:

```
class Test<T>
{
    T obj;
    //...



}
```

Пример:

```
class Test<T>
{
    T obj;
    //...
    obj = null; // подходит только для ссылочных типов
    obj = 0; // подходит только для числовых типов и
             // перечислений, но не для структур
}
```

Пример:

```
class Test<T>
{
    T obj;
    //...
    obj = null; // подходит только для ссылочных типов
    obj = 0; // подходит только для числовых типов и
             // перечислений, но не для структур
}
```

Пример:

```
class Test<T>
{
    T obj;
    //...
    obj = null; // подходит только для ссылочных типов
    obj = 0; // подходит только для числовых типов и
             // перечислений, но не для структур
}
```

Пример:

```
class Test<T>
{
    T obj;
    //...
  • obj = null; // подходит только для ссылочных типов
  • obj = 0; // подходит только для числовых типов и
             // перечислений, но не для структур
}
```

Значения по умолчанию переменной параметра типа

```
class имя_класса<T>
{
    T obj;
    public Test()
    {
      obj = default(T);
    }
}
```

Пример:

```
class Test<T>
{
    T obj;
    //...
  • obj = null; // подходит только для ссылочных типов
  • obj = 0; // подходит только для числовых типов и
             // перечислений, но не для структур
}
```

Значения по умолчанию переменной параметра типа

```
class имя_класса<T>
{
    T obj;
    public Test()
    {
      obj = default(T);
    }
}
```

Упражнение 11.4

Продемонстрировать применение оператора default.

Общий вид

```
struct имя_структуры<параметры> {
```

Общий вид

```
struct имя_структуры<параметры> {
```

Ограничения

```
struct имя_структуры<T> where T : Ограничение { // ...
```

Общий вид

```
struct имя_структуры<параметры> {
```

Ограничения

```
struct имя_структуры<T> where T : Ограничение { // ...
```

Упражнение 11.5

В одном из предыдущих упражнение на обобщения заменить обобщенный класс - структурой.

Обобщенный метод

```
class имя_класса
{
    тип имя_метода<список_параметров>(параметры_метода) {... }
}
```

Обобщенный метод

```
class имя_класса
{
    тип имя_метода<список_параметров>(параметры_метода) {... }
}
```

Пример:

```
public static bool CopyInsert<T>(T[ ] arrayBase, T[ ] arrayTarget)
```

Обобщенный метод

```
class имя_класса
{
    тип имя_метода<список_параметров>(параметры_метода) {... }
}
```

Пример:

```
public static bool CopyInsert<T>(T[ ] arrayBase, T[ ] arrayTarget)
```

Замечание

Отметим два момента касающихся обобщенного метода. Параметр типа объявляется после имени метода, но перед списком его параметров. Обобщенный метод вызывается также как и обычный: без указания аргументов типа.

Обобщенный метод

```
class имя_класса
{
    тип имя_метода<список_параметров>(параметры_метода) {... }
}
```

Пример:

```
public static bool CopyInsert<T>(T[ ] arrayBase, T[ ] arrayTarget)
```

Замечание

Отметим два момента касающихся обобщенного метода. Параметр типа объявляется после имени метода, но перед списком его параметров. Обобщенный метод вызывается также как и обычный: без указания аргументов типа.

Упражнение 11.6

Написать обобщенный статический метод для копирования элементов из одного массива в другой.

Общая форма объявления обобщенного делегата

```
delegate возвращаемый_тип
имя_делегата<список_параметров_типа>(список_аргументов);
```

Общая форма объявления обобщенного делегата

delegate возвращаемый_тип
имя_делегата<список_параметров_типа>(список_аргументов);

Пример:

```
delegate T SomeOp<T>(T v);
```

Общая форма объявления обобщенного делегата

```
delegate возвращаемый_тип
имя_делегата<список_параметров_типа>(список_аргументов);
```

Пример:

```
delegate T SomeOp<T>(T v);
```

Замечание

Отметим, что тип T может служить в качестве возвращаемого типа, несмотря на то, что параметр типа T указывается после имени делегата SomeOp.

Общая форма объявления обобщенного делегата

delegate возвращаемый_тип
имя_делегата<список_параметров_типа>(список_аргументов);

Пример:
delegate T SomeOp<T>(T v);

Замечание

Отметим, что тип T может служить в качестве возвращаемого типа, несмотря на то, что параметр типа T указывается после имени делегата SomeOp.

Упражнение 11.7

Реализовать пример обобщенного делегата.

Объявление обобщенного интерфейса

interface имя_интерфейса<список_параметров_типа>

Объявление обобщенного интерфейса

interface имя_интерфейса<список_параметров_типа>

Объявление класса, реализующего обобщенный интерфейс

class имя_класса<список_параметров_типа> :
имя_интерфейса<список_параметров_типа>

Объявление обобщенного интерфейса

interface имя_интерфейса<список_параметров_типа>

Объявление класса, реализующего обобщенный интерфейс

class имя_класса<список_параметров_типа> :
имя_интерфейса<список_параметров_типа>

Пример:

```
public interface ISeries<T> {...}
class ByTwos<T> : ISeries<T> {...}
```

Объявление обобщенного интерфейса

interface имя_интерфейса<список_параметров_типа>

Объявление класса, реализующего обобщенный интерфейс

class имя_класса<список_параметров_типа> :
имя_интерфейса<список_параметров_типа>

Пример:

```
public interface ISeries<T> {...}
class ByTwos<T> : ISeries<T> {...}
```

Упражнение 11.8

Переделать упражнения 6.1 и 6.2 используя обобщенный интерфейс. В методе GetNext() использовать делегат(что позволит выбирать метод получения следующего члена ряда при создании экземпляра класса). Сами методы определить в отдельном классе.

Пример:

```
//Не годится!
public static bool IsIn<T>(T what, T[ ] obs)
{
    foreach (T v in obs)
        if (v == what) //Ошибка!
            return true;
        return false;
}
```

Пример:

```
//Не годится!
public static bool IsIn<T>(T what, T[ ] obs)
{
    foreach (T v in obs)
        if (v == what) //Ошибка!
            return true;
        return false;
}
```

Замечание

Два объекта параметров обобщенного типа должны реализовывать интерфейс IComparable или IComparable<T> и/или интерфейс IEquatable<T> для того чтобы их можно было сравнивать. В обоих вариантах интерфейса IComparable для этой цели определен метод CompareTo(), а в интерфейсе IEquatable<T> — метод Equals().

Пример:

```
//Не годится!
public static bool IsIn<T>(T what, T[ ] obs)
{
    foreach (T v in obs)
        if (v == what) //Ошибка!
            return true;
        return false;
}
```

Замечание

Два объекта параметров обобщенного типа должны реализовывать интерфейс IComparable или IComparable<T> и/или интерфейс IEquatable<T> для того чтобы их можно было сравнивать. В обоих вариантах интерфейса IComparable для этой цели определен метод CompareTo(), а в интерфейсе IEquatable<T> — метод Equals().

Форма объявления интерфейса IEquatable<T>

```
public interface IEquatable<T>
```

Пример:

```
//Требуется обобщенный интерфейс IEquatable<T>.
public static bool IsIn<T>(T what, T[ ] obs) where T : IEquatable<T>
{
    foreach(T v in obs)
        if(v.Equals(what)) //Применяется метод Equals().
            return true;
        return false;
}
```

Замечание

Два объекта параметров обобщенного типа должны реализовывать интерфейс IComparable или IComparable<T> и/или интерфейс IEquatable<T> для того чтобы их можно было сравнивать. В обоих вариантах интерфейса IComparable для этой цели определен метод CompareTo(), а в интерфейсе IEquatable<T> — метод Equals().

Форма объявления интерфейса IEquatable<T>

```
public interface IEquatable<T>
```

Форма объявления интерфейса IComparable<T>

```
public interface IComparable<T>
```

Форма объявления интерфейса IComparable<T>

```
public interface IComparable<T>
```

Метод CompareTo()

```
int CompareTo(T other)
```

Метод возвращает нуль, если вызывающий объект оказывается равен объекту other; положительное значение, если больше и отрицательное - если вызывающий объект оказывается меньше, чем объект other.

Форма объявления интерфейса IComparable<T>

```
public interface IComparable<T>
```

Метод CompareTo()

int CompareTo(T other)
Метод возвращает нуль, если вызывающий объект оказывается равен объекту other; положительное значение, если больше и отрицательное - если вызывающий объект оказывается меньше, чем объект other.

Пример:

```
//Требуется обобщенный интерфейс IComparable<T>. В данном методе
//предполагается, что массив отсортирован. Он возвращает значение
// true, если значение параметра what оказывается среди элементов
//массива, передаваемых параметру obs.
public static bool InRange<T>(T what, T[ ] obs) where T : IComparable<T>
{
   if(what.CompareTo(obs[0]) < 0 || what.CompareTo(obs[obs.Length-1]) > 0)
       return false;
   return true;
}
```

Форма объявления интерфейса IComparable<T>

```
public interface IComparable<T>
```

Метод CompareTo()

int CompareTo(T other)
Метод возвращает нуль, если вызывающий объект оказывается равен объекту other; положительное значение, если больше и отрицательное - если вызывающий объект оказывается меньше, чем объект other.

Упражнение 11.9

Продемонстрировать применение обобщенных интерфейсов IComparable<T> и IEquatable<T>, используя методы InRange и IsIn. Рассмотреть два массива: один из элементов типа int, второй - из объектов пользовательского типа MyClass

Иерархия обобщенных классов

```
class класс_2<T> : класс_1<T>
```

Иерархия обобщенных классов

```
class класс_2<T> : класс_1<T>
```

Замечание

Параметр типа T указывается в объявлении класса класс_2 и в то же время передается классу класс_1. Это означает, что любой тип, передаваемый классу класс_2, будет передаваться также классу класс_1.

Иерархия обобщенных классов

```
class класс_2<T> : класс_1<T>
```

Замечание

Параметр типа T указывается в объявлении класса класс_2 и в то же время передается классу класс_1. Это означает, что любой тип, передаваемый классу класс_2, будет передаваться также классу класс_1.

Упражнение 11.10

Реализовать обобщенный производный класс с двумя параметрами наследующий от обобщенного базового с одним параметром. В оба класса добавить конструкторы с непустым набором параметров типа.

Иерархия обобщенных классов

```
class класс_2<T> : класс_1<T>
```

Замечание

Параметр типа T указывается в объявлении класса класс_2 и в то же время передается классу класс_1. Это означает, что любой тип, передаваемый классу класс_2, будет передаваться также классу класс_1.

Упражнение 11.10

Реализовать обобщенный производный класс с двумя параметрами наследующий от обобщенного базового с одним параметром. В оба класса добавить конструкторы с непустым набором параметров типа.

Упражнение 11.11

Реализовать обобщенный производный класс от необобщенного базового

Пример:

```
//В этом обобщенном интерфейсе поддерживается ковариантность.
public interface IMyCoVarGenIF<out T>{ T GetObject(); }
//Реализовать интерфейс IMyCoVarGenIF.
class MyClass<T> : IMyCoVarGenIF<T>
{
    T obj;
    public MyClass(T v) { obj = v; }
    public T GetObject() { return obj; }
}
//Создать простую иерархию классов.
class Alpha
{
    string name;
    public Alpha(string n) { name = n; }
    public string GetName() { return name; }
}
class Beta : Alpha
{
    public Beta(string n) : base(n) { }
}
```

```
//Создать ссылку из интерфейса IMyCoVarGenIF на объект типа
MyClass<Alpha>.
//Это вполне допустимо как при наличии ковариантности, так и без нее.
IMyCoVarGenIF<Alpha> AlphaRef = new MyClass<Alpha>(new
Alpha("Alpha #1"));
Console.WriteLine("Имя объекта, на который ссылается переменная
AlphaRef: " + AlphaRef.GetObject().GetName());
//А теперь создать объект MyClass<Beta> и присвоить его переменной
AlphaRef.
//*** Эта строка кода вполне допустима благодаря ковариантности. ***
AlphaRef = new MyClass<Beta>(new Beta("Beta #1"));
Console.WriteLine("Имя объекта, на который теперь ссылается " +
"переменная AlphaRef: " + AlphaRef.GetObject().GetName());
```

```
//Создать ссылку из интерфейса IMyCoVarGenIF на объект типа
MyClass<Alpha>.
//Это вполне допустимо как при наличии ковариантности, так и без нее.
IMyCoVarGenIF<Alpha> AlphaRef = new MyClass<Alpha>(new
Alpha("Alpha #1"));
Console.WriteLine("Имя объекта, на который ссылается переменная
AlphaRef: " + AlphaRef.GetObject().GetName());
//А теперь создать объект MyClass<Beta> и присвоить его переменной
AlphaRef.
//*** Эта строка кода вполне допустима благодаря ковариантности. ***
AlphaRef = new MyClass<Beta>(new Beta("Beta #1"));
Console.WriteLine("Имя объекта, на который теперь ссылается " +
"переменная AlphaRef: " + AlphaRef.GetObject().GetName());
```

Обобщенный интерфейс IMyContraVarGenIF контравариантного типа.

```
public interface IMyContraVarGenIF<in T>
```

Упражнение 11.12

Реализовать пример переопределения виртуального метода в обобщенном классе.

Упражнение 11.12

Реализовать пример переопределения виртуального метода в обобщенном классе.

Упражнение 11.13

Перегрузка методов с параметрами типа может привести к неоднозначности. Продемонстрировать это на примере.

Упражнение 11.12

Реализовать пример переопределения виртуального метода в обобщенном классе.

Упражнение 11.13

Перегрузка методов с параметрами типа может привести к неоднозначности. Продемонстрировать это на примере.

Упражнение 11.14

Продемонстрировать ковариантность и контравариантность в обобщенном интерфейсе.

Упражнение 11.12

Реализовать пример переопределения виртуального метода в обобщенном классе.

Упражнение 11.13

Перегрузка методов с параметрами типа может привести к неоднозначности. Продемонстрировать это на примере.

Упражнение 11.14

Продемонстрировать ковариантность и контравариантность в обобщенном интерфейсе.

Упражнение 11.15

Продемонстрировать конвариантность и контравариантность в обобщенных делегатах.