Иногда возникает потребность сравнить два экземпляра параметра типа. Допустим,

что требуется написать обобщенный метод IsIn(), возвращающий логическое

значение true, если в массиве содержится некоторое значение. Для этой цели сначала

можно попробовать сделать следующее.

// Не годится!

public static bool IsIn<T>(T what, T[] obs)

{

foreach (T v in obs)

if (v == what) // Ошибка!

return true;

return false;

}

К сожалению, эта попытка не пройдет. Ведь параметр Т относится к обобщенному

типу, и поэтому компилятору не удастся выяснить, как сравнивать два объекта. Требуется

ли для этого поразрядное сравнение или же только сравнение отдельных полей?

А возможно, сравнение ссылок? Вряд ли компилятор сможет найти ответы на эти вопросы.

Правда, из этого положения все же имеется выход.

Для сравнения двух объектов параметра обобщенного типа они должны реализовывать

интерфейс IComparable или IComparable<T> и/или интерфейс IEquatable<T>.

В обоих вариантах интерфейса IComparable для этой цели определен метод

CompareTo(), а в интерфейсе IEquatable<T> — метод Equals(). Разновидности

интерфейса IComparable предназначены для применения в тех случаях, когда требуется

определить относительный порядок следования двух объектов. А интерфейс

IEquatable служит для определения равенства двух объектов. Все эти интерфейсы

определены в пространстве имен System и реализованы во встроенных в C# типах данных,

включая int, string и double. Но их нетрудно реализовать и для собственных

создаваемых классов. Итак, начнем с обобщенного интерфейса IEquatable<T>.

Интерфейс IEquatable<T> объявляется следующим образом.

public interface IEquatable<T>

Сравниваемый тип данных передается ему в качестве аргумента типа Т. В этом интерфейсе

определяется метод Equals(), как показано ниже.

bool Equals(Т other)

В этом методе сравниваются взывающий объект и другой объект, определяемый

параметром *other.* В итоге возвращается логическое значение true, если оба объекта

равны, а иначе — логическое значение false.

В ходе реализации интерфейса IEquatable<T> обычно требуется также переопределять

методы GetHashCode() и Equals(Object), определенные в классе Object,

чтобы они оказались совместимыми с конкретной реализацией метода Equals().

Ниже приведен пример программы, в которой демонстрируется исправленный вариант

упоминавшегося ранее метода IsIn().

// 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;

}

Обратите внимание в приведенном выше примере на применение следующего

ограничения.

where T : IEquatable<T>

Это ограничение гарантирует, что только те типы, в которых реализован интерфейс

IEquatable, являются действительными аргументами типа для метода IsIn(). Внутри этого метода применяется метод Equals(), который определяет равенство одного

объекта другому.

Для определения относительного порядка следования двух элементов применяется

интерфейс IComparable. У этого интерфейса имеются две формы: обобщенная и необобщенная.

Обобщенная форма данного интерфейса обладает преимуществом обеспечения

типовой безопасности, и поэтому мы рассмотрим здесь именно ее. Обобщенный

интерфейс IComparable<T> объявляется следующим образом.

public interface IComparable<T>

Сравниваемый тип данных передается ему в качестве аргумента типа Т. В этом интерфейсе

определяется метод CompareTo(), как показано ниже.

int CompareTo(Т other)

В этом методе сравниваются вызывающий объект и другой объект, определяемый

параметром *other.* В итоге возвращается нуль, если вызывающий объект оказывается

больше, чем объект *other;* и отрицательное значение, если вызывающий объект оказывается

меньше, чем объект *other.*

Для того чтобы воспользоваться методом CompareTo(), необходимо указать ограничение,

которое требуется наложить на аргумент типа для реализации обобщенного

интерфейса IComparable<T>. А затем достаточно вызвать метод CompareTo(), чтобы

сравнить два экземпляра параметра типа.

Ниже приведен пример применения обобщенного интерфейса IComparable<T>.

В этом примере вызывается метод InRange(), возвращающий логическое значение

true, если объект оказывается среди элементов отсортированного массива.

// Требуется обобщенный интерфейс IComparable<T>. В данном методе

// предполагается, что массив отсортирован. Он возвращает логическое

// 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;

}

В приведенном ниже примере программы демонстрируется применение обоих

методов IsIn() и InRange() на практике.

(***glava18\_16***)

//now in MyClass realised generalized

//interfaces IComparable<T> and IEquatable<T>

class MyClass : IComparable<MyClass>, IEquatable<MyClass>

{

public int Val;

public MyClass(int x) { Val = x; }

//realize Generalized interface IComparable<T>

public int CompareTo(MyClass other)

{

return Val - other.Val; //Now, no cast is needed

}

//realize generalized interface IEquatable<T>

public bool Equals(MyClass other)

{

return Val == other.Val;

}

//override method Equals(Object)

public override bool Equals(object obj)

{

if (obj is MyClass)

return Equals((MyClass)obj);

return false;

}

//override method GetHashCode()

public override int GetHashCode()

{

return Val.GetHashCode();

}

}

class CompareDemo

{

//demands generalized interface IEquatable<T>

public static bool IsIn<T>(T what, T[] obs) where T : IEquatable<T>

{

foreach (T v in obs)

if (v.Equals(what))

return true;

return false;

}

//demands generalized interface IComparable<T>

//in this method assume array is sorted. It returns logic

//value true, if value of what is in array 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;

}

//demonstrate

static void Main()

{

//use IsIn() method to int type

int[] nums = { 1, 2, 3, 4, 5 };

if (IsIn(2, nums))

Console.WriteLine("Value 2 has been found.");

if (IsIn(99, nums))

Console.WriteLine("Wount output");

//use IsIn method to MyClass

MyClass[] mcs = { new MyClass(1), new MyClass(2),

new MyClass(3), new MyClass(4)};

if (IsIn(new MyClass(3), mcs))

Console.WriteLine("MyClass(3) object has been found.");

if (IsIn(new MyClass(99), mcs))

Console.WriteLine("Wount output.");

//use InRange() to int type

if (InRange(2, nums))

Console.WriteLine("Value 2 in borders of nums array");

if (InRange(1, nums))

Console.WriteLine("Value 1 in borders of nums array");

if (InRange(5, nums))

Console.WriteLine("Value 5 in borders of nums array");

if (!InRange(0, nums))

Console.WriteLine("Value 0 is NOT in borders of nums array");

if (!InRange(6, nums))

Console.WriteLine("Value 6 in NOT in borders of nums array");

//use method InRange() to MyClass objects

if (InRange(new MyClass(2), mcs))

Console.WriteLine("Object of MyClass(2), in borders of array nums.");

if (InRange(new MyClass(1), mcs))

Console.WriteLine("Object of MyClass(1), in borders of array nums.");

if (InRange(new MyClass(4), mcs))

Console.WriteLine("Object of MyClass(4), in borders of array nums.");

if (!InRange(new MyClass(0), mcs))

Console.WriteLine("Object of MyClass(0), NOT in borders of array nums.");

if (!InRange(new MyClass(5), mcs))

Console.WriteLine("Object of MyClass(5), NOT in borders of array nums.");

}

}

