Класс BitArray служит для хранения отдельных битов в коллекции. А поскольку

в коллекции этого класса хранятся биты, а не объекты, то своими возможностями он

отличается от классов других коллекций. Тем не менее в классе BitArray реализуются

интерфейсы ICollection и IEnumerable как основополагающие элементы поддержки

всех типов коллекций. Кроме того, в классе BitArray реализуется интерфейс

ICloneable.

В классе BitArray определено несколько конструкторов. Так, с помощью приведенного

ниже конструктора можно сконструировать объект типа BitArray из массива

логических значений.

public BitArray (bool [] values)

В данном случае каждый элемент массива *values* становится отдельным битом в

коллекции. Это означает, что каждому элементу массива values соответствует отдельный

бит в коллекции. Более того, порядок расположения элементов в массиве values

сохраняется и в коллекции соответствующих им битов.

Коллекцию типа BitArray можно также составить из массива байтов, используя

следующий конструктор.

public BitArray(byte[] bytes)

Здесь битами в коллекции становится уже целый их набор из массива *bytes,* причем

элемент *bytes*[0] обозначает первые 8 битов, элемент *bytes*[1] — вторые 8 битов

и т.д. Аналогично, коллекцию типа BitArray можно составить из массива целочисленных

значений, используя приведенный ниже конструктор.

public BitArray(int[] values)

В данном случае элемент values[0] обозначает первые 32 бита, элемент

*values*[1] — вторые 32 бита и т.д.

С помощью следующего конструктора можно составить коллекцию типа BitArray,

указав ее конкретный размер:

public BitArray(int length)

где *length* обозначает количество битов в коллекции, которые инициализируются

логическим значением false. В приведенном ниже конструкторе можно указать не

только размер коллекции, но и первоначальное значение составляющих ее битов.

public BitArray(int length, bool defaultValue)

В данном случае все биты в коллекции инициализируются значением

*defaultValue,* передаваемым конструктору в качестве параметра.

И наконец, новую коллекцию типа BitArray можно создать из уже существующей,

используя следующий конструктор.

public BitArray(BitArray bits)

Вновь сконструированный объект будет содержать такое же количество битов, как

и в указываемой коллекции *bits*, а в остальном это будут две совершенно разные коллекции.

Коллекции типа BitArray подлежат индексированию. По каждому индексу указывается

отдельный бит в коллекции, причем нулевой индекс обозначает младший бит.

В классе BitArray определяется ряд собственных методов, помимо тех, что уже

объявлены в интерфейсах, которые в нем реализуются. Методы этого класса приведены

в табл. 25.9. Обратите внимание на то, что в классе BitArray не поддерживается

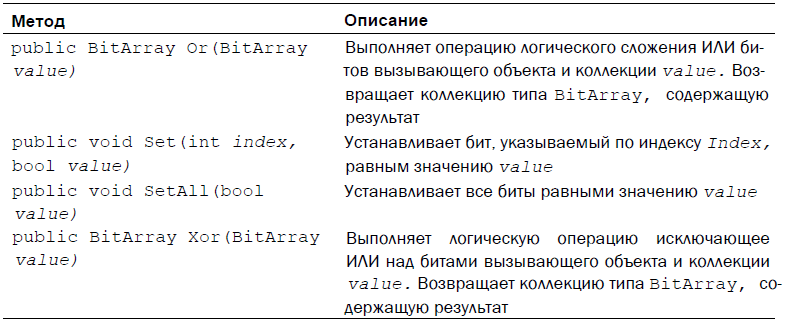
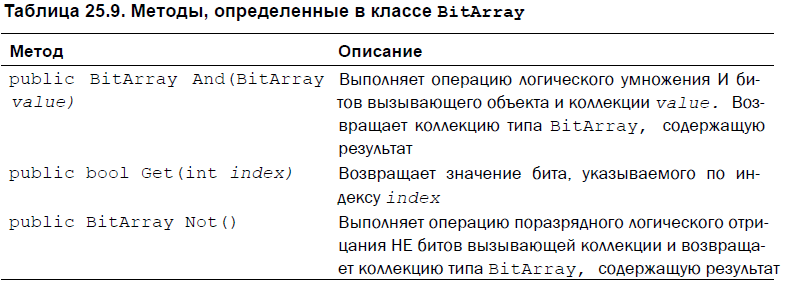
метод Synchronized(). Это означает, что для коллекций данного класса синхронизированная

оболочка недоступна, а свойство IsSynchronized всегда имеет логическое

значение false. Тем не менее для управления доступом к коллекции типа BitArray

ее можно синхронизировать для объекта, предоставляемого упоминавшимся ранее

свойством SyncRoot.



В классе BitArray определяется также собственное свойство, помимо тех, что указаны

в интерфейсах, которые в нем реализуются.

public int Length { get; set; }

Свойство Length позволяет установить или получить количество битов в коллекции.

Следовательно, оно возвращает такое же значение, как и стандартное свойство

Count, определяемое для всех коллекций. В отличие от свойства Count, свойство

Length доступно не только для чтения, но и для записи, а значит, с его помощью можно

изменить размер коллекции типа BitArray. Так, при сокращении коллекции типа

BitArray лишние биты усекаются, начиная со старшего разряда. А при расширении

коллекции типа BitArray дополнительные биты, имеющие логическое значение

false, вводятся в коллекцию, начиная с того же старшего разряда.

Кроме того, в классе BitArray определяется следующий индексатор.

public bool this[int index] { get; set; }

С помощью этого индексатора можно получать или устанавливать значение элемента.

В приведенном ниже примере демонстрируется применение класса BitArray.

(***glava25\_8***)

class BADemo

{

public static void ShowBits(string rem,

BitArray bits)

{

Console.WriteLine(rem);

for (int i = 0; i < bits.Count; i++)

Console.Write("{0, -6} ", bits[i]);

Console.WriteLine("\n");

}

static void Main()

{

BitArray ba = new BitArray(8);

byte[] b = { 67 };

BitArray ba2 = new BitArray(b);

ShowBits("Content of ba: ", ba);

ba = ba.Not();

ShowBits("ba after Not(): ", ba);

ShowBits("Content of ba2: ", ba2);

BitArray ba3 = ba.Xor(ba2);

ShowBits("Result of ba XOR ba2: ", ba3);

}

}