Класс Array относится к числу наиболее часто используемых в пространстве имен

System. Он является базовым классом для всех массивов в С#. Следовательно, его методы

можно применять к массивам любого встроенного в C# типа или же к массивам

определяемого пользователем типа. Свойства, определенные в классе Array, перечислены

в табл. 21.11, а методы — в табл. 21.12.

В классе Array реализуются следующие интерфейсы: ICloneable, ICollection,

IEnumerable, IStructuralComparable, IStructuralEquatable, а также IList.

Все интерфейсы, кроме ICloneable, определены в пространстве имен System.

Collections, подробнее рассматриваемом в главе 25.

В ряде методов данного класса используется параметр типа IComparer или

IComparer<T>. Интерфейс IComparer находится в пространстве имен System.

Collections. В нем определяется метод Compare() для сравнения значений двух

объектов, как показано ниже.

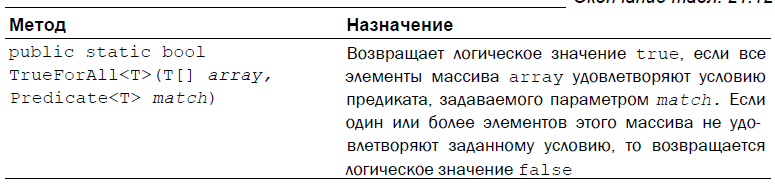
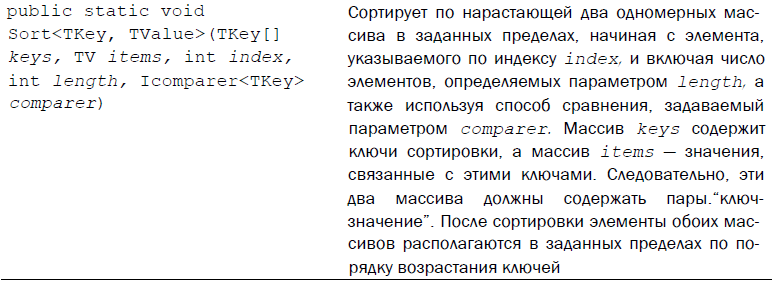
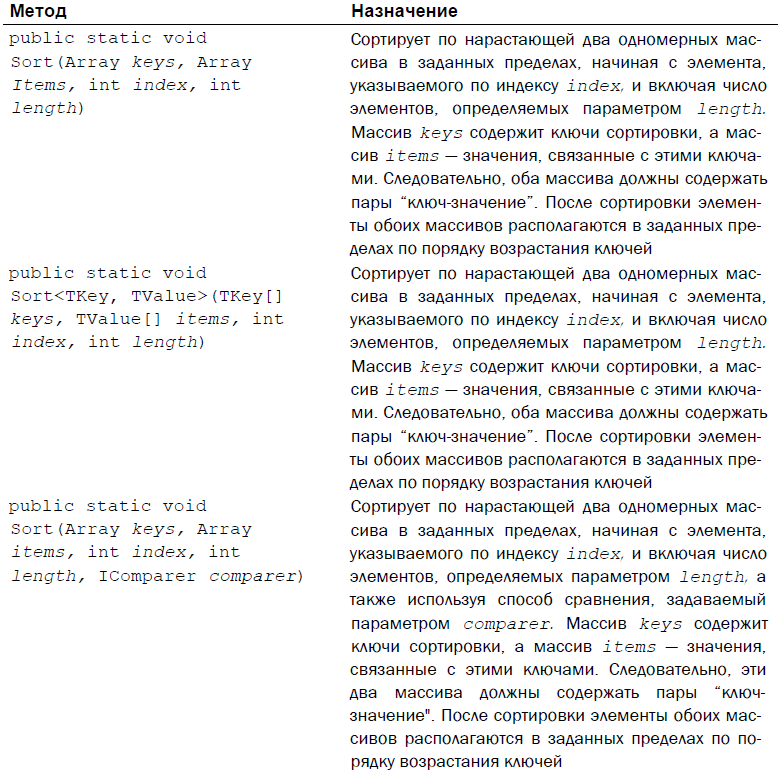
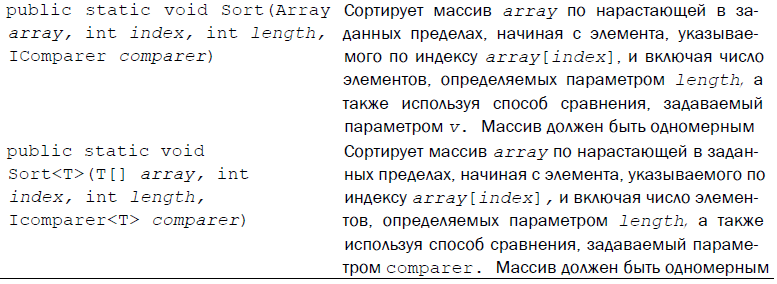
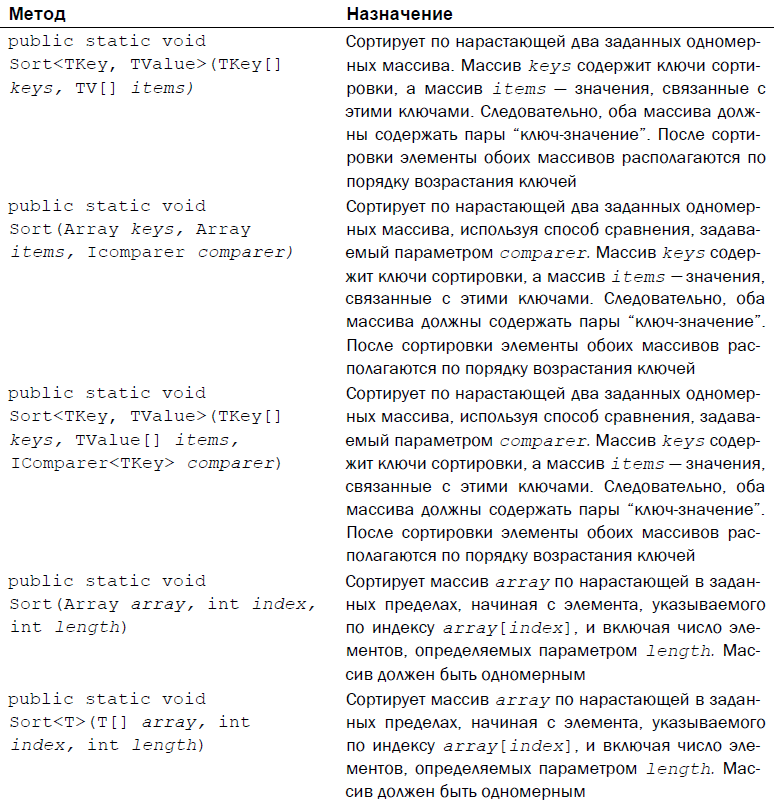
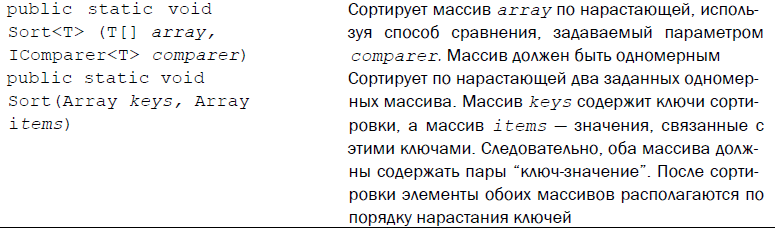
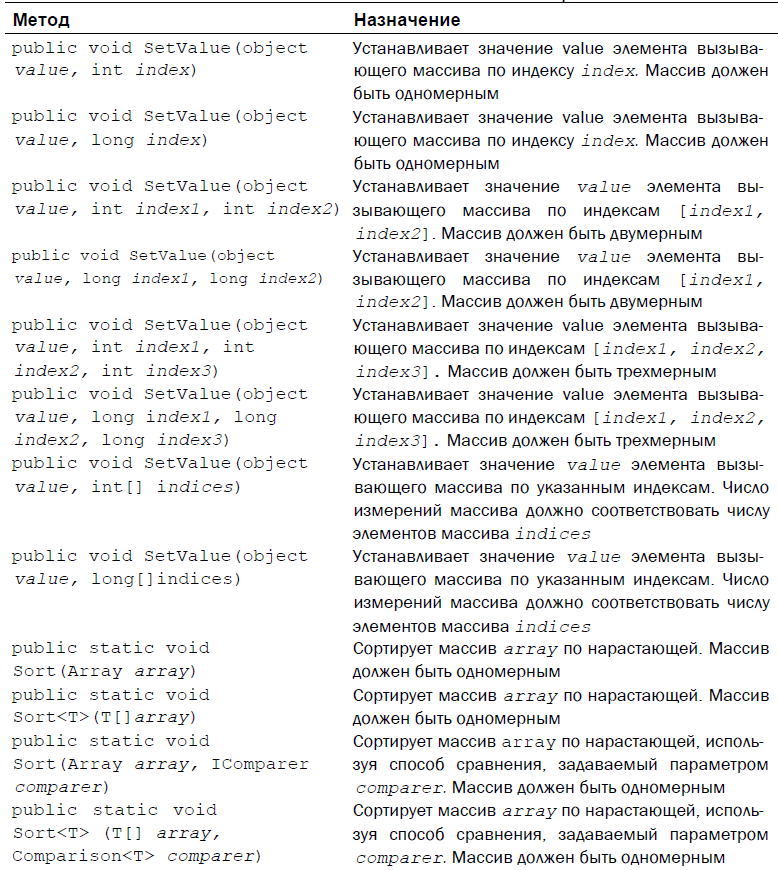
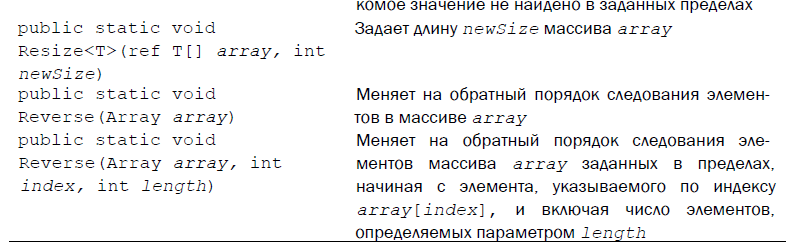
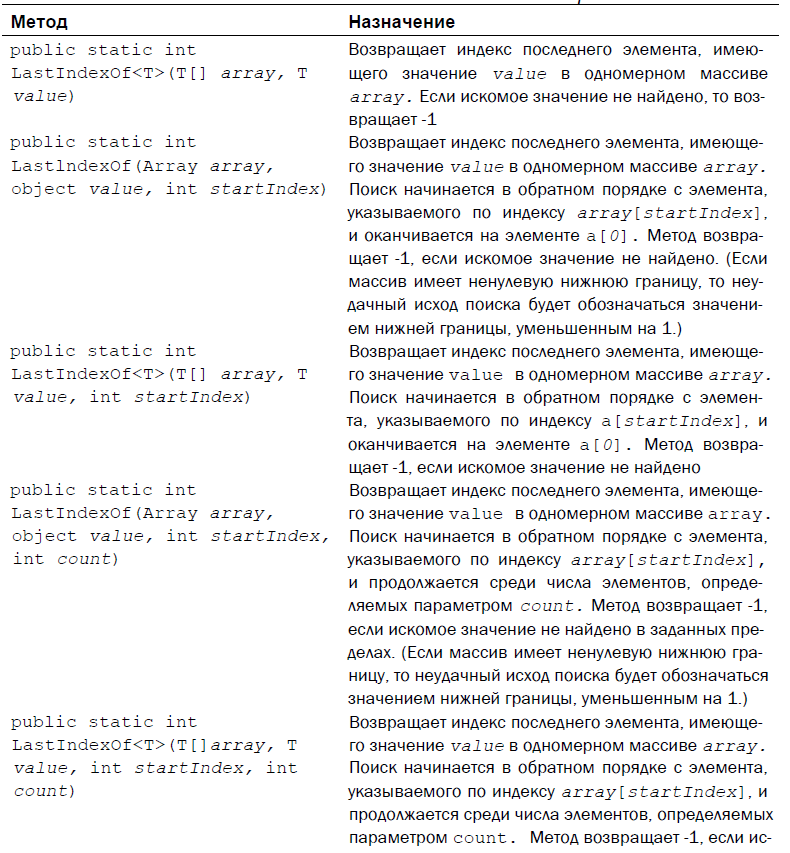
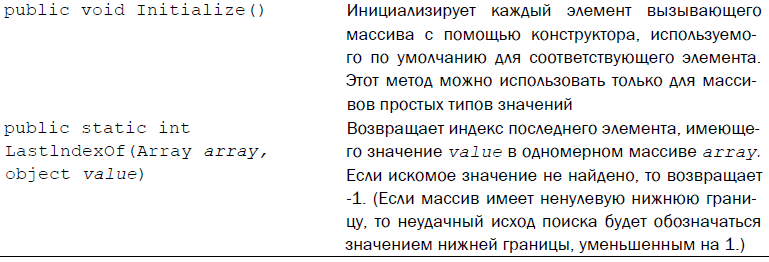
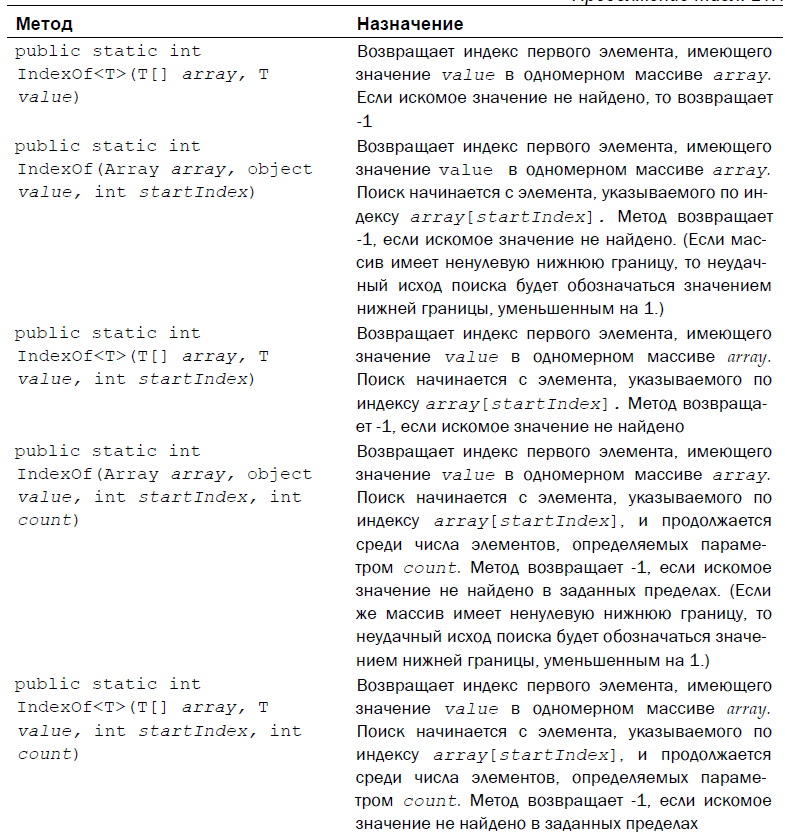
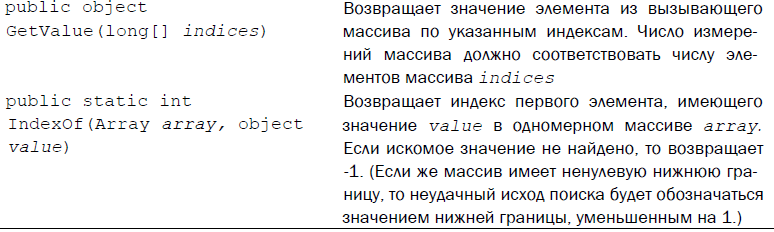
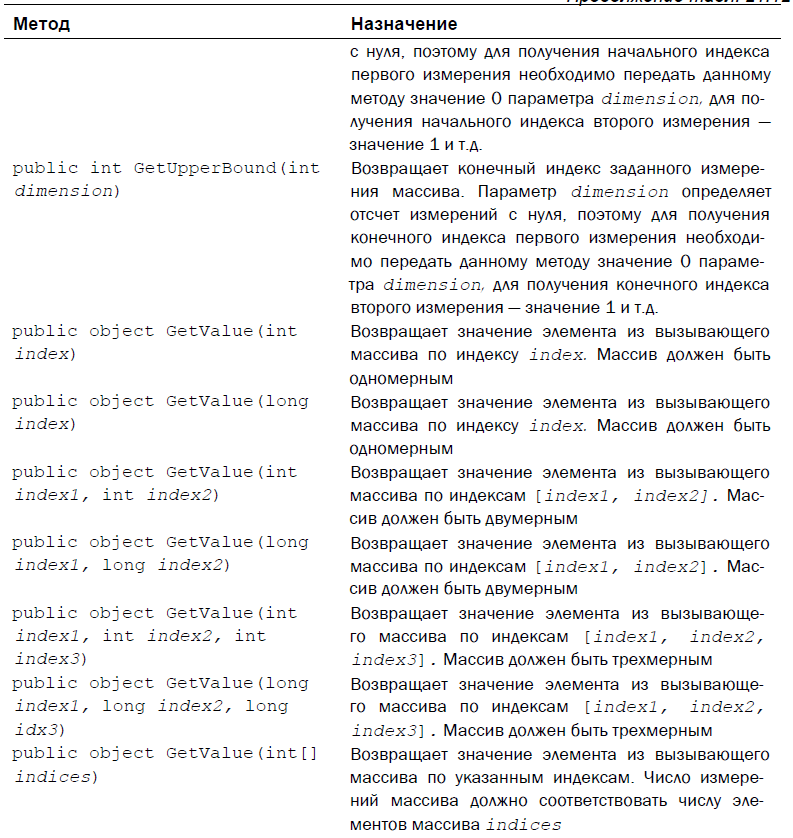
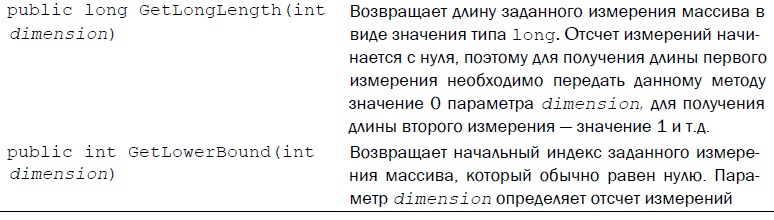
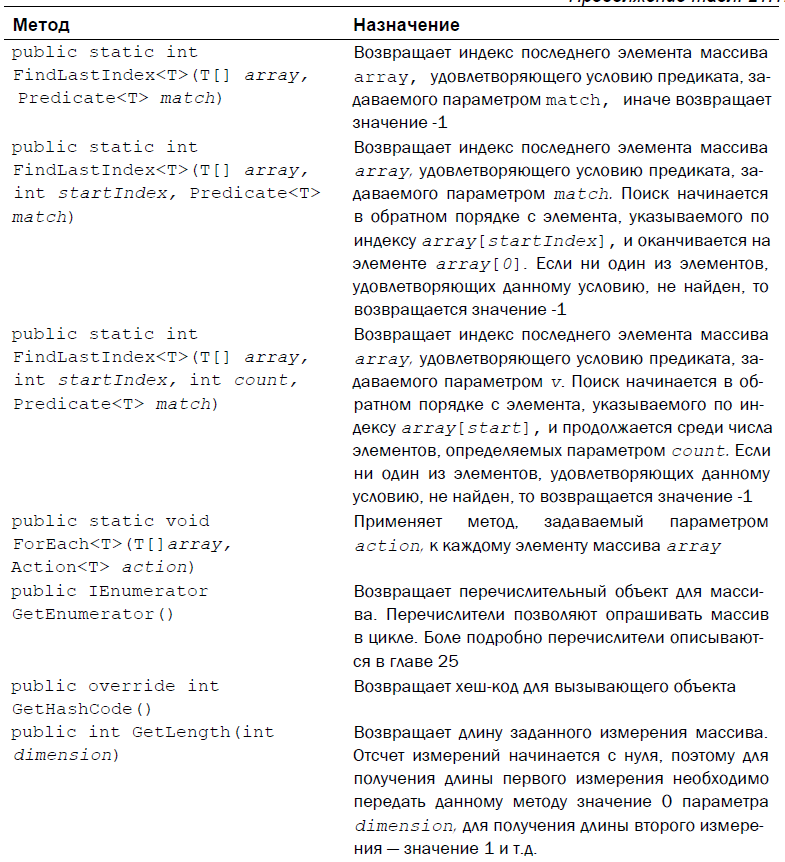
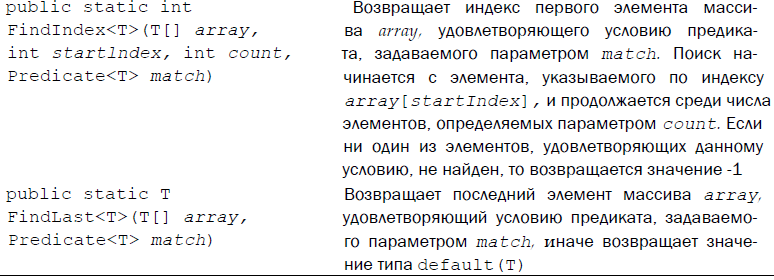
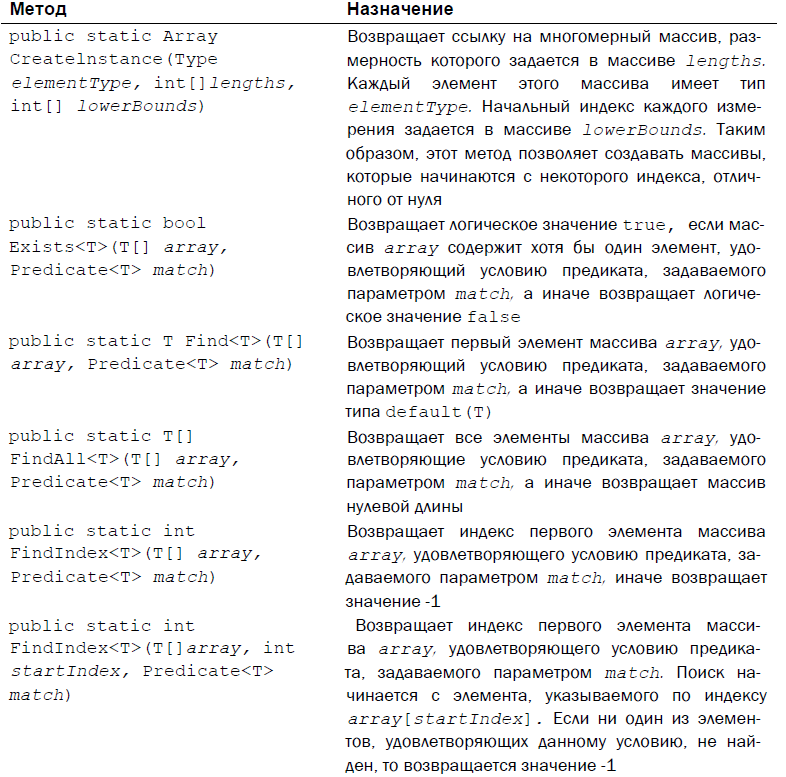
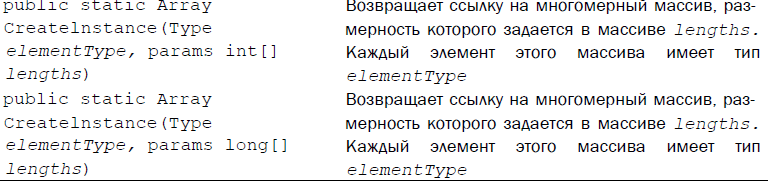
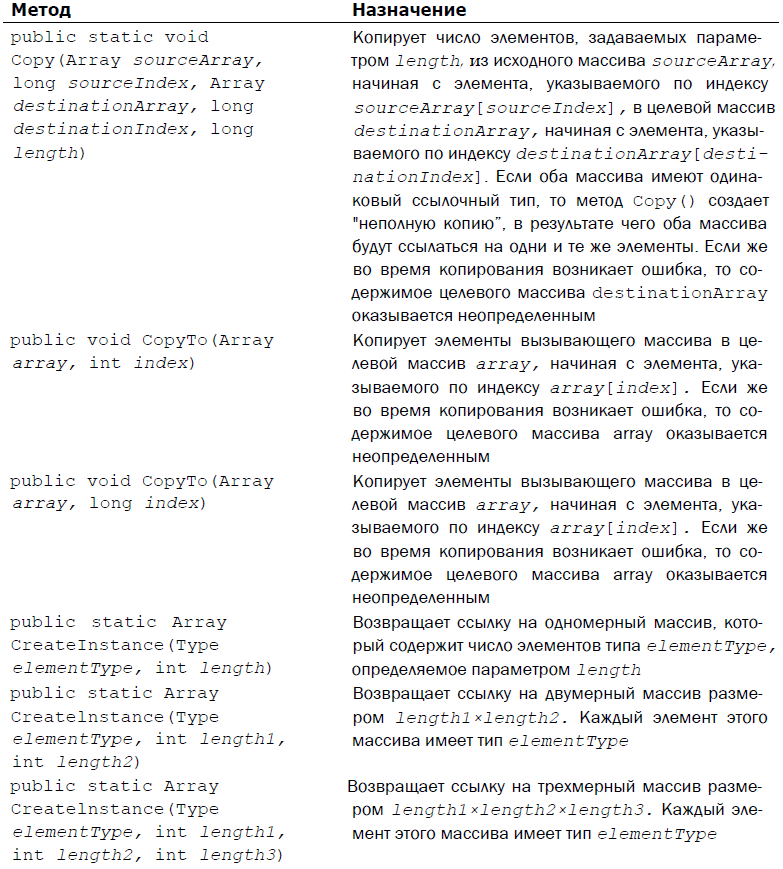
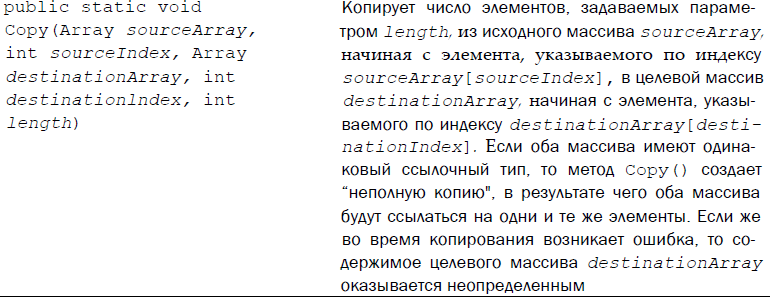
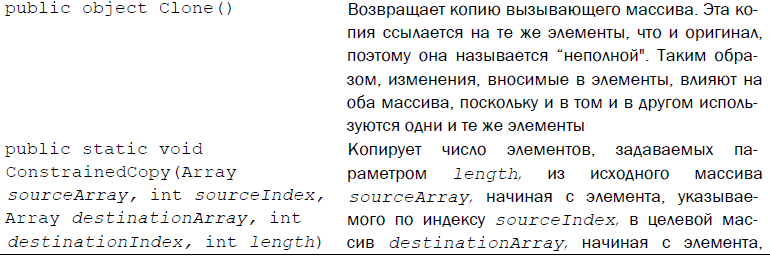
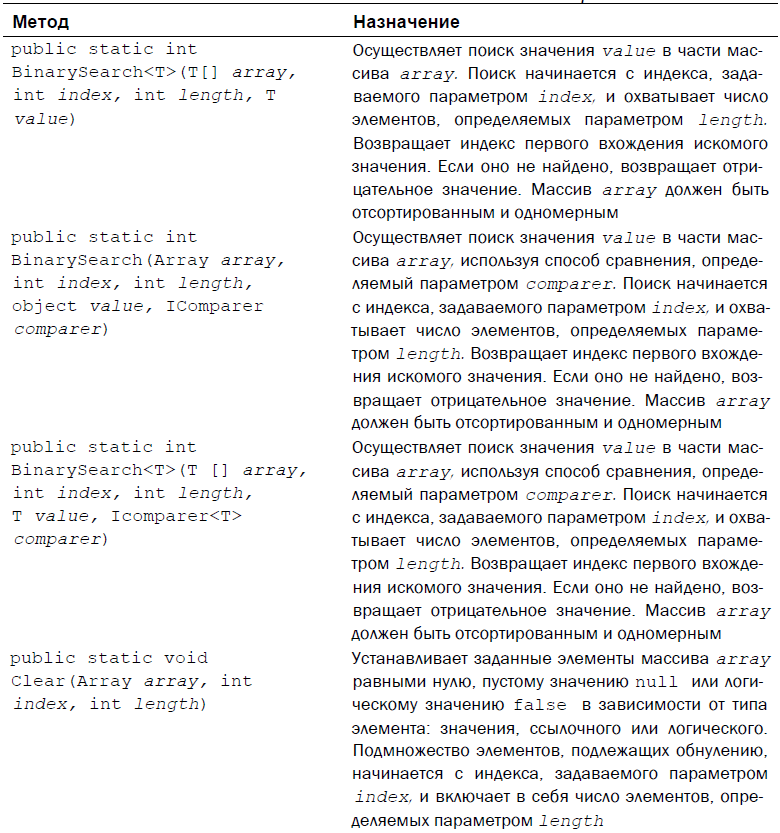
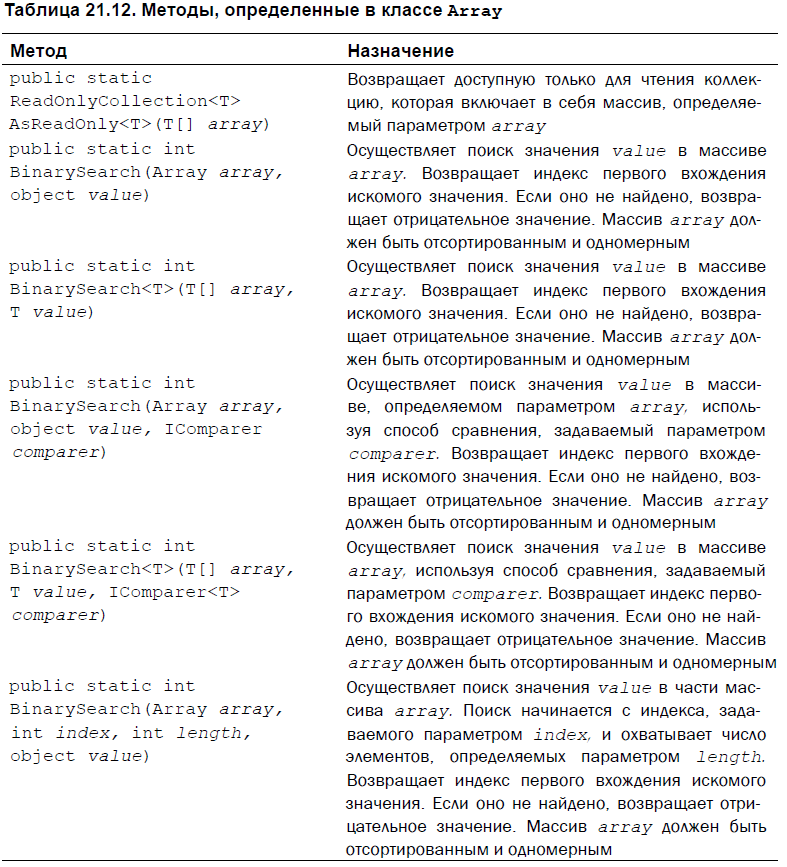
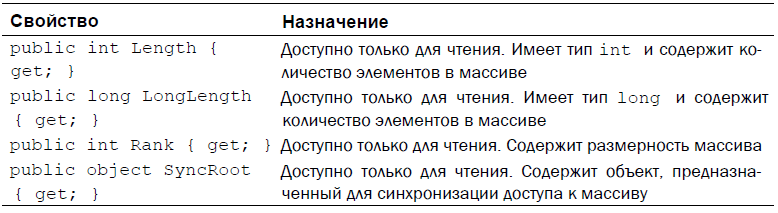
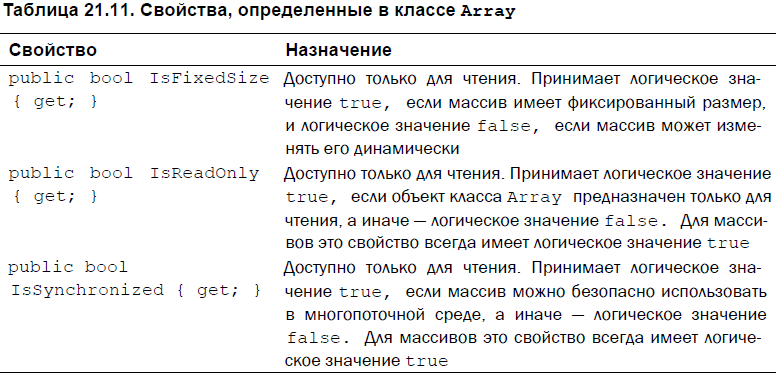
int Compare(object х, object у)

Этот метод возвращает значение больше нуля, если х больше у; значение меньше

нудя, если *х* меньше *у*; и, наконец, нулевое значение, если оба значения равны.

В последующих разделах демонстрируется ряд наиболее распространенных операций

с массивами.



**Сортировка и поиск в массивах**

Содержимое массива нередко приходится сортировать. Для этой цели в классе

Array предусмотрен обширный ряд сортирующих методов. Так, с помощью разных

вариантов метода Sort() можно отсортировать массив полностью или в заданных

пределах либо отсортировать два массива, содержащих соответствующие пары

"ключ-значение". После сортировки в массиве можно осуществить эффективный поиск,

используя разные варианты метода BinarySearch(). В качестве примера ниже

приведена программа, в которой демонстрируется применение методов Sort()

и BinarySearch() для сортировки и поиска в массиве значений типа int.

(***glava21\_3***)

class SortDemo

{

static void Main()

{

int[] nums = { 5, 4, 6, 3, 14, 9, 8, 17, 1, 24, -1, 0 };

//show nums

Console.Write("Order of nums: ");

foreach (var i in nums)

Console.Write(i + " ");

Console.WriteLine();

//sort array

Array.Sort(nums);

//Show after sort

Console.Write("Order of nums after sort: ");

foreach (var i in nums)

Console.Write(i + " ");

Console.WriteLine();

//find 14

int idx = Array.BinarySearch(nums, 14);

Console.WriteLine("Index of element of value 14 in array nums: " + idx);

}

}

Все методы, определенные в классе Array, автоматически

доступны для обработки массивов всех встроенных в C# типов значений.

Но в отношении массивов ссылок на объекты это правило может и не соблюдаться.

Так, для сортировки массива ссылок на объекты в классе типа этих объектов должен

быть реализован интерфейс IComparable или IComparable<T>. Если же ни один из

этих интерфейсов не реализован в данном классе, то во время выполнения программы

может возникнуть исключительная ситуация в связи с попыткой отсортировать подобный

массив или осуществить в нем поиск. Правда, реализовать оба интерфейса,

IComparable и IComparable<T>, совсем нетрудно.

В интерфейсе IComparable определяется один метод.

int CompareTo(object obj)

В этом методе значение вызывающего объекта сравнивается со значением объекта,

определяемого параметром *obj.* Если значение вызывающего объекта больше, чем

у объекта *obj*, то возвращается положительное значение; если оба значения равны —

нулевое значение, а если значение вызывающего объекта меньше, чем у объекта *obj, —*

отрицательное значение.

Интерфейс IComparable<T> является обобщенным вариантом интерфейса

IComparable. Поэтому в нем определен следующий обобщенный вариант метода

CompareTo().

int CompareTo(Т other)

Обобщенный вариант метода CompareTo() действует аналогично необобщенному

его варианту. В нем значение вызывающего объекта также сравнивается со значением

объекта, определяемого параметром *other.* Если значение вызывающего объекта больше,

чем у объекта *other,* то возвращается положительное значение; если оба значения

равны — нулевое значение, а если значение вызывающего объекта меньше, чем у объекта

*other,* — отрицательное значение.

В качестве примера ниже приведена программа, в которой демонстрируются сортировка

и поиск в массиве объектов определяемого пользователем класса.

(***glava21\_4***)

class MyClass : IComparable<MyClass>

{

public int i;

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

//realize interface IComparable<>

public int CompareTo(MyClass v)

{

return i - v.i;

}

public bool Eqauls(MyClass v)

{

return i == v.i;

}

}

class SortDemo

{

static void Main()

{

MyClass[] nums = new MyClass[5];

nums[0] = new MyClass(5);

nums[1] = new MyClass(2);

nums[2] = new MyClass(3);

nums[3] = new MyClass(4);

nums[4] = new MyClass(1);

//show order

Console.Write("Order of nums: ");

foreach (var i in nums)

Console.Write(i.i + " ");

Console.WriteLine();

//sort nums

Array.Sort(nums);

//show order

Console.Write("Order of nums after sort: ");

foreach (var i in nums)

Console.Write(i.i + " ");

Console.WriteLine();

//find MyClass(2)

MyClass x = new MyClass(2);

int indx = Array.BinarySearch(nums, x);

Console.WriteLine("Index of MyClass(2): " + indx);

}

}

Класс StringComparer объявляется в пространстве имен System и реализует,

среди прочего, интерфейсы IComparer и IComparer<T>. Поэтому экземпляр объекта

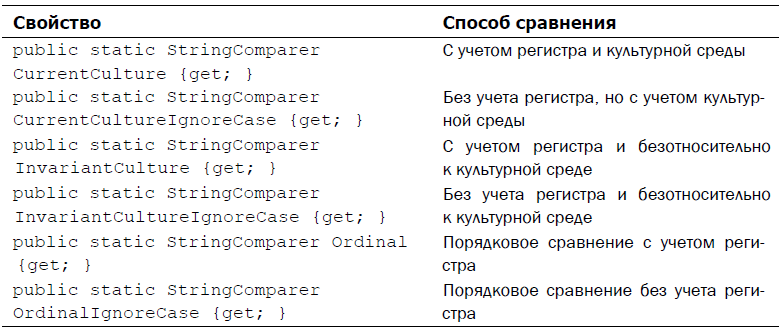
типа StringComparer может быть передан в качестве аргумента параметру типа

IComparer. Кроме того, в классе StringComparer определен ряд доступных только

для чтения свойств, возвращающих экземпляр объекта типа StringComparer и поддерживающих

различные способы сравнения символьных строк. Все эти свойства перечислены

ниже.



Передавая явным образом экземпляр объекта типа StringComparer, можно совершенно

однозначно определить порядок сортировки или поиска в массиве. Например,

в приведенном фрагменте кода сортировка и поиск в массиве символьных строк

осуществляется с помощью свойства StringComparer.Ordinal.

string[] strs = { "xyz", "one", "beta", "Alpha" };

// ...

Array.Sort(strs, StringComparer.Ordinal);

int idx = Array.BinarySearch(strs, "beta", StringComparer.Ordinal);

**Обращение содержимого массива**

**(glava21\_5)**

class ReverseDemo

{

static void Main()

{

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

//show order

Console.Write("Order of nums: ");

foreach (var i in nums)

Console.Write(i + " ");

Console.WriteLine();

//sort nums

Array.Reverse(nums);

//show order

Console.Write("Order of nums after sort: ");

foreach (var i in nums)

Console.Write(i + " ");

Console.WriteLine();

//reverse part of array

Array.Reverse(nums, 1, 3);

Console.Write("Order of nums after sort: ");

foreach (var i in nums)

Console.Write(i + " ");

Console.WriteLine();

}

}

**Копирование массива**

Полное или частичное копирование одного массива в другой — это еще одна весьма

распространенная операция с массивами. Для копирования содержимого массива

служит метод Сору(). В зависимости от его варианта копирование элементов исходного

массива осуществляется в начало или в средину целевого массива. Применение

метода Сору() демонстрируется в приведенном ниже примере программы.

(***glava21\_6***)

class CopyDemo

{

static void Main()

{

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

int[] target = { 11, 12, 13, 14, 15 };

int[] source2 = { -1, -2, -3, -4, -5 };

//show order

Console.Write("Source: ");

foreach (var i in source)

Console.Write(i + " ");

Console.WriteLine();

//show target

Console.Write("Target: ");

foreach (var i in target)

Console.Write(i + " ");

Console.WriteLine();

//copy array

Array.Copy(source, target, source.Length);

//show target

Console.Write("Target after copying: ");

foreach (var i in target)

Console.Write(i + " ");

Console.WriteLine();

//copy into midle of target

Array.Copy(source2, 2, target, 3, 2);

//show target

Console.Write("Target after partial copying: ");

foreach (var i in target)

Console.Write(i + " ");

Console.WriteLine();

}

}

**Применение предиката**

*Предикат* представляет собой делегат типа System.Predicate, возвращающий

логическое значение true иди false в зависимости от некоторого условия. Он объявляется

следующим образом.

public delegate bool Predicate<T>(T obj)

Объект, проверяемый по заданному условию, передается в качестве параметра

*obj.* Если объект *obj* удовлетворяет заданному условию, то предикат должен возвратить

логическое значение true, в противном случае — логическое значение false.

Предикаты используются в ряде методов класса Array, включая: Exists(), Find(),

FindIndex() и FindAll().

В приведенном ниже примере программы демонстрируется применение предиката

с целью определить, содержится ли в целочисленном массиве отрицательное значение.

Если такое значение обнаруживается, то данная программа извлекает первое

отрицательное значение, найденное в массиве. Для этого в ней используются методы

Exists() и Find().

(***glava21\_7***)

class PredDemo

{

//predicate method returns true, if

//value of v is negative

static bool IsNeg(int v)

{

if (v < 0) return true;

return false;

}

static void Main()

{

int[] nums = { 1, 3, -1, 5, -9 };

Console.Write("Nums: ");

foreach (var i in nums)

Console.Write(i + " ");

Console.WriteLine();

//check if negative in nums

if (Array.Exists(nums, PredDemo.IsNeg))

{

Console.WriteLine("Nums has negative value.");

//find first negative value in array

int x = Array.Find(nums, PredDemo.IsNeg);

Console.WriteLine("First negative value: " + x);

}

else

Console.WriteLine("Nums does not have negative values.");

}

}

В данном примере программы в качестве предиката методам Exists() и Find()

передается метод IsNeg(). Обратите внимание на следующее объявление метода

IsNeg().

static bool IsNeg(int v)

{

if (v < 0) return true;

return false;

}

Методы Exists() и Find() автоматически и по порядку передают элементы массива

переменной v. Следовательно, после каждого вызова метода IsNeg() переменная

v будет содержать следующий элемент массива.

**Применение делегата Action**

Делегат Action применяется в методе Array.ForEach() для выполнения заданного

действия над каждым элементом массива. Существуют разные формы делегата Action,

отличающиеся числом параметров типа. Ниже приведена одна из таких форм.

public delegate void Action<T>(T obj)

В этой форме объект, над которым должно выполняться действие, передается в качестве

параметра *obj.* Когда же эта форма делегата Action применяется в методе

Array.ForEach(), то каждый элемент массива передается по порядку объекту *obj.*

Следовательно, используя делегат Action и метод ForEach(), можно в одном операторе

выполнить заданную операцию над целым массивом.

В приведенном ниже примере программы демонстрируется применение делегата

Action и метода ForEach(). Сначала в ней создается массив объектов класса MyClass,

а затем используется метод Show() для отображения значений, извлекаемых из этого

массива. Далее эти значения становятся отрицательными с помощью метода Neg().

И наконец, метод Show() используется еще раз для отображения отрицательных значений.

Все эти операции выполняются посредством вызовов метода ForEach().

(***glava21\_8***)

class MyClass

{

public int i;

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

}

class ActionDemo

{

//method of delegate Action, shows value it takes

static void Show(MyClass o)

{

Console.Write(o.i + " ");

}

//one more method delegate Action

//makes negative value it takes

static void Neg(MyClass o)

{

o.i = -o.i;

}

static void Main()

{

MyClass[] nums = new MyClass[5];

nums[0] = new MyClass(5);

nums[1] = new MyClass(2);

nums[2] = new MyClass(3);

nums[3] = new MyClass(4);

nums[4] = new MyClass(1);

Console.Write("Nums: ");

//make action to see values

Array.ForEach(nums, ActionDemo.Show);

Console.WriteLine();

//make Action do negative values

Array.ForEach(nums, ActionDemo.Neg);

Console.Write("Nums array after negative: ");

Array.ForEach(nums, ActionDemo.Show);

}

}