**Класс Hashtable**

Класс Hashtable предназначен для создания коллекции, в которой для хранения

ее элементов служит хеш-таблица. Как должно быть известно большинству читателей,

информация сохраняется в *хеш-таблице* с помощью механизма, называемого *хешированием.*

При хешировании для определения уникального значения, называемого *хеш-*

*кодом,* используется информационное содержимое специального ключа. Подученный

в итоге хеш-код служит в качестве индекса, по которому в таблице хранятся искомые

данные, соответствующие заданному ключу. Преобразование ключа в хеш-код выполняется

автоматически, и поэтому сам хеш-код вообще недоступен пользователю. Преимущество

хеширования заключается в том, что оно обеспечивает постоянство времени

выполнения операций поиска, извлечения и установки значений независимо от величины

массивов данных. В классе Hashtable реализуются интерфейсы IDictionary,

ICollection, IEnumerable, ISerializable, IDeserializationCallback и

ICloneable.

В классе Hashtable определено немало конструкторов. Ниже приведены наиболее

часто используемые конструкторы этого класса.

public Hashtable ()

public Hashtable (IDictionary d)

public Hashtable (int capacity)

public Hashtable (int capacity,

float loadFactor)

В первой форме создается создаваемый по умолчанию объект класса Hashtable.

Во второй форме создаваемый объект типа Hashtable инициализируется элементами

из коллекции *d*. В третьей форме создаваемый объект типа Hashtable инициализируется,

учитывая емкость коллекции, задаваемую параметром *capacity.* И в четвертой

форме создаваемый объект типа Hashtable инициализируется, учитывая заданную

емкость *capacity* и коэффициент заполнения *loadFactor.* Коэффициент заполнения,

иногда еще называемый *коэффициентом загрузки,* должен находиться в пределах

от 0,1 до 1,0. Он определяет степень заполнения хеш-таблицы до увеличения ее размера.

В частности, таблица расширяется, если количество элементов оказывается больше

емкости таблицы, умноженной на коэффициент заполнения. В тех конструкторах, которые

не принимают коэффициент заполнения в качестве параметра, этот коэффициент

по умолчанию выбирается равным 1,0.

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

объявлены в интерфейсах, которые в нем реализуются. Некоторые из наиболее часто

используемых методов этого класса приведены в табл. 25.5. В частности, для того чтобы

определить, содержится ли ключ в коллекции типа Hashtable, вызывается метод

ContainsKey(). А для того чтобы выяснить, хранится ли в такой коллекции конкретное

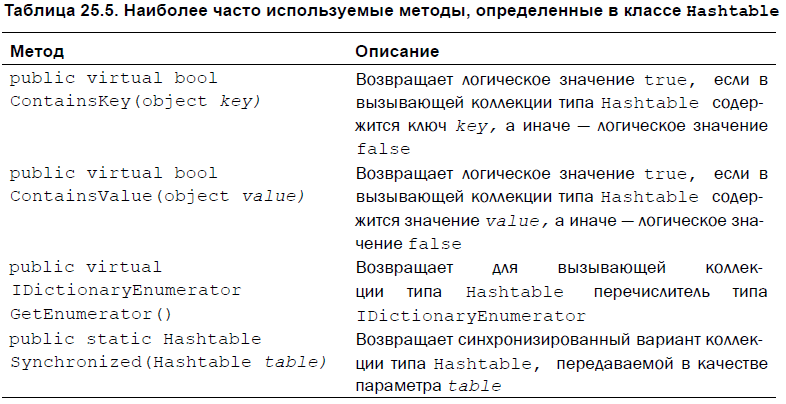
значение, вызывается метод ContainsValue(). Для перечисления содержимого

коллекции типа Hashtable служит метод GetEnumerator(), возвращающий объект

типа IDictionaryEnumerator. Напомним, что IDictionaryEnumerator — это

перечислитель, используемый для перечисления содержимого коллекции, в которой

хранятся пары "ключ-значение".



В классе Hashtable доступны также открытые свойства, определенные в тех интерфейсах,

которые в нем реализуются. Особая роль принадлежит двум свойствам,

Keys и Values, поскольку с их помощью можно получить ключи или значения из

коллекции типа Hashtable. Эти свойства определяются в интерфейсе IDictionary

следующим образом.

public virtual ICollection Keys { get; }

public virtual ICollection Values { get; }

В классе Hashtable не поддерживаются упорядоченные коллекции, и поэтому

ключи или значения получаются из коллекции в произвольном порядке. Кроме того,

в классе Hashtable имеется защищенное свойство EqualityComparer. А два других

свойства, hcp и comparer, считаются устаревшими.

Пары "ключ-значение" сохраняются в коллекции типа Hashtable в форме структуры

типа DictionaryEntry, но чаще всего это делается без прямого вмешательства со стороны пользователя, поскольку свойства и методы оперируют ключами и значениями

по отдельности. Если, например, в коллекцию типа Hashtable добавляется

элемент, то для этой цели вызывается метод Add(), принимающий два аргумента:

ключ и значение.

Нужно, однако, иметь в виду, что сохранение порядка следования элементов в коллекции

типа Hashtable не гарантируется. Дело в том, что процесс хеширования оказывается,

как правило, непригодным для создания отсортированных таблиц.

Ниже приведен пример программы, в которой демонстрируется применение класса

Hashtable.

(***glava25\_4***)

class HashTableDemo

{

static void Main()

{

//create hash table

Hashtable ht = new Hashtable();

//add elements

ht.Add("buildings", "living area");

ht.Add("car", "transport");

ht.Add("book", "collection of words");

ht.Add("apple", "fruit");

//add elements through indexator

ht["tractor"] = "agrarian machine";

//get keys collection

ICollection c = ht.Keys;

//use keys to get values

foreach(var str in c)

Console.WriteLine(str + ": " + ht[str]);

}

}

Как следует из приведенного выше результата, пары "ключ-значение" сохраняются

в произвольном порядке. Обратите внимание на то, как получено и отображено содержимое

хеш-таблицы ht. Сначала была получена коллекция ключей с помощью свойства

Keys. Затем каждый ключ был использован для индексирования хеш-таблицы ht

с целью извлечь из нее значение, соответствующее заданному ключу. Напомним, что

в качестве индекса в данном случае использовался индексатор, определенный в интерфейсе

IDictionary и реализованный в классе Hashtable.