

КЛАССЫ ОБОБЩЕННЫХ КОЛЛЕКЦИЙ

ЛЕКЦИЯ 14



ОБОБЩЕННЫЕ КОЛЛЕКЦИИ

- Пространство имен `System.Collections.Generic` содержит большое число классов обобщенных коллекций
- На основе интерфейсов, реализуемых классами коллекций, они могут быть разделены на:
 - ✓ списки,
 - ✓ собственно коллекции,
 - ✓ словари

ОБОБЩЕННЫЕ ИНТЕРФЕЙСЫ

Интерфейс	Описание
<code>IEnumerable<T></code>	Необходим, когда с коллекцией используется оператор <code>foreach</code> ; определяет метод <code>GetEnumerator()</code> , возвращающий перечислитель, который реализует интерфейс <code>IEnumerator</code>
<code>ICollection<T></code>	Интерфейс, реализуемый классами обобщенных коллекций
<code>IList<T></code>	Этот интерфейс определяет индексатор, поэтому предназначен для создания списков, элементы которых доступны по своим позициям. Унаследован от интерфейса <code>ICollection<T></code>
<code>ISet<T></code>	Интерфейс, реализуемый множествами. Унаследован от интерфейса <code>ICollection<T></code>

ОБОБЩЕННЫЕ ИНТЕРФЕЙСЫ

Интерфейс	Описание
IDictionary<TKey, TValue>	Реализуется обобщенными классами коллекций, элементы которых состоят из ключа и значения
ILookup<TKey, TValue>	Подобно IDictionary<TKey, TValue> поддерживает ключи и значения. Однако в этом случае коллекция может содержать множественные значения для одного ключа
ICompare<T>	Интерфейс IComparer<T> используется для сортировки элементов внутри коллекции с помощью метода Compare ()
IEqualityComparer<T>	Этот интерфейс позволяет проверять объекты на предмет эквивалентности друг другу

СПИСКИ

- Обобщенный класс `List<T>` предназначен для работы с динамическими списками
- Этот класс реализует необобщенные интерфейсы `IList`, `ICollection`, `IEnumerable`, а также аналогичные обобщенные интерфейсы `IList<T>`, `ICollection<T>` и `IEnumerable<T>`

ЕМКОСТЬ И РАЗМЕР СПИСКА

- *Емкостью списка* называется число элементов, которые потенциально могут быть размещены в пределах выделенной для него памяти
- *Размером списка* называется число реально содержащихся в нем элементов
- Размер списка всегда меньше или равен его емкости и может быть получен с помощью свойства для чтения `Count`

СОЗДАНИЕ СПИСКА

- Класс `List<T>` имеет три конструктора:
 - ✓ **public** `List<T>()` – конструктор по умолчанию, создающий пустой список с нулевой емкостью;
 - ✓ **public** `List<T>(Int32)` – конструктор, создающий список с заданной начальной емкостью;
 - ✓ **public** `List<T>(IEnumerable<T>)` – конструктор, создающий список и копирующий в нее элементы указанной коллекции
- В последнем случае аргументом конструктора может быть любая существующая коллекция с типом элементов `T`, в том числе, массив

ЗАПОЛНЕНИЕ СПИСКА

- Производится путем последовательного добавления в него новых элементов в конец списка
- Если список создан с помощью конструктора по умолчанию, то после добавления в него первого элемента его емкость устанавливается равной 4
- Если список создан с помощью копирующего конструктора, то его емкость и размер устанавливаются равными размеру копируемой коллекции

ЗАПОЛНЕНИЕ СПИСКА

- По мере заполнения списка его размер приближается к его емкости
- Когда размер списка становится равным его емкости последняя увеличивается вдвое; при этом создается новый список, в который копируются элементы существующего списка
- Для добавления новых элементов класс `List<T>` имеет два метода:
 - ✓ **public void** `Add(T)` – добавляет один элемент в конец списка;
 - ✓ **public void** `AddRange(IEnumerable<T>)` – добавляет в конец списка элементы указанной коллекции

ИНИЦИАЛИЗАЦИЯ СПИСКА

- Первоначальное заполнение списка можно выполнять с помощью инициализаторов коллекций
- Синтаксис инициализаторов коллекций подобен инициализаторам массивов, а именно, инициализатор имеет вид списка значений элементов, заключенного в фигурные скобки:
`var intList = new List < int > () {1, 2}; //выведение типа`
`var stringList = new List < string > () {"one", "two"};`

ДОБАВЛЕНИЕ В ЗАДАННУЮ ПОЗИЦИЮ

- Для добавления нового элемента в произвольное место списка класс `List<T>` имеет два метода, аналогичным методам добавления элементов в конец списка:
 - ✓ **public void** `Insert(int, T)`,
 - ✓ **public void** `Insert(int, IEnumerable<T>)`
- Во втором случае индекс, указывающий место добавления элементов в список не должно превышать размера списка

ДОСТУП К ЭЛЕМЕНТАМ

- Все классы, реализующие интерфейсы `IList` и `IList<T>`, предоставляют индексатор
- Поэтому к элементам списка можно обращаться с использованием индексатора, передавая ему номер элемента
- Первый элемент доступен по индексу 0
- Отметим, что индексный доступ в классах коллекций возможен для коллекций `ArrayList`, `StringCollection` и `List<T>`

ДОСТУП К ЭЛЕМЕНТАМ

- Поскольку `List<T>` реализует интерфейс `IEnumerable`, проход по элементам коллекции можно также осуществлять с помощью оператора `foreach`
- Вместо оператора `foreach` в классе `List<T>` также предусмотрен метод `ForEach (Action<T>)`, объявленный с параметром `Action<T>`:

```
public void ForEach(Action<T>);
```

где `Action<T>` - параметр-делегат

ДЕЛЕГАТЫ

- Делегаты в языке C# – это аналоги указателей функций языка C++
- Причины замены указателей функций делегатами:
 1. C# запрещает использование указателей в безопасном коде;
 2. C# является полностью объектно-ориентированным языком, а указатели – это простые переменные, а не объекты

ОБЪЯВЛЕНИЕ УКАЗАТЕЛЕЙ ФУНКЦИЙ

- В отличие от обычных указателей указатели функций связываются не только с типом, которым для них является тип возвращаемого функцией значения, но и с ее сигнатурой
- Поэтому указатели функций в C++ объявляются следующим образом:
 <тип_результата> (*<имя_указателя>)
 (<сигнатура>)

ОБЪЯВЛЕНИЕ УКАЗАТЕЛЕЙ ФУНКЦИЙ

□ Например:

```
double (*fptr) (int, double)
```

является указателем на функцию с двумя параметрами целого и вещественного типов, которая возвращает значение вещественного типа

□ Указателю функции можно присвоить адрес точки входа любой функции с типом и сигнатурой этого указателя

УКАЗАТЕЛИ ФУНКЦИЙ

□ Например:

```
double myFunc (int, double);
```

```
double x, y; int n;
```

```
cin >> n >> x;
```

```
fptr = myFunc;
```

```
y = (*fptr) (n, x);
```

□ Вызов функции myFunc производится через указатель fptr

ОБЪЯВЛЕНИЕ ДЕЛЕГАТА

- Делегаты являются объектами – экземплярами класса, создаваемого средой .Net на основании объявления класса делегата, которое имеет вид:

```
delegate <тип_результата> <имя_класса_делегата>  
(<список формальных параметров>);
```

ОБЪЯВЛЕНИЕ ДЕЛЕГАТА

- Следует обратить внимание на то, что при объявлении класса делегата указывается не сигнатура, а список формальных параметров (типы и имена формальных параметров)

ПРИМЕНЕНИЕ ДЕЛЕГАТА

- Как и указатели функций экземпляры класса-делегата (делегаты) получают значения через присваивание имен методов
- Для статических методов дополнительно указывается имя содержащего их класса, для нестатических – экземпляра класса
- Пример применения делегата
- Тестирование примера

ДОСТУП К ЭЛЕМЕНТАМ

□ Этот делегат объявлен в классе `List<T>` следующим образом:

```
public delegate void Action<T> (T x)
```

и определяет метод, который применяется для обработки очередного элемента коллекции

□ Аргументом при вызове метода `ForEach` может служить имя любого метода, соответствующего объявлению этого делегата

УДАЛЕНИЕ ЭЛЕМЕНТОВ

□ Элементы списка можно удалять двумя способами:

- ✓ с указанием значения удаляемого элемента,
- ✓ с указанием индекса удаляемого элемента

□ В первом случае используется метод
`public bool Remove (T),`

возвращающий значение `true` при удачном завершении операции и `false`, если указанное значение в списке не найдено

УДАЛЕНИЕ ЭЛЕМЕНТОВ

- Во втором случае используется метод:
`public void RemoveAt(int)`
- Метод `RemoveRange (int, int)` удаляет множество элементов из коллекции
- Первый параметр специфицирует индекс, начиная с которого располагаются удаляемые элементы, а второй параметр задает количество удаляемых элементов
- Удаление по индексу работает быстрее, поскольку в этом случае не приходится выполнять поиск удаляемого элемента по всему списку

УДАЛЕНИЕ ЭЛЕМЕНТОВ

- Чтобы удалить из коллекции все элементы, удовлетворяющие некоторому условию нужно использовать метод `RemoveAll (Predicate<T>)`
- Этот метод имеет параметр-делегат `Predicate<T>`, определяющий условия удаления элементов
- Для удаления всех элементов из коллекции служит метод `Clear()`

ПОИСК ЭЛЕМЕНТОВ

- Для поиска элементов в списке можно использовать целый ряд методов
- Одним из параметров всех этих методов является делегат `Predicate<T>` , объявленный следующим образом:
public delegate bool Predicate<T> (T x)
- Это позволяет указывать в качестве условия поиска имя любого метода, возвращающего булевское значение и имеющего один параметр типа `T`

МЕТОДЫ ПОИСКА

Метод	Описание
<code>public T Find (Predicate<T>)</code>	Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает первое вхождение в пределах всего списка
<code>public List<T> FindAll (Predicate<T>)</code>	Выполняет поиск всех элементов, удовлетворяющего условиям указанного предиката, и возвращает их список
<code>public int FindIndex (Predicate<T>)</code>	Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс первого вхождения в пределах всего списка
<code>public int FindLastIndex (Predicate<T>)</code>	Выполняет поиск элемента, удовлетворяющего условиям указанного предиката, и возвращает отсчитываемый от нуля индекс последнего вхождения в пределах всего списка

ПОИСК ЭЛЕМЕНТОВ

- Еще одним методом поиска является метод **public int** IndexOf (T)
- Этот метод имеет в качестве параметра элемент списка и возвращает индекс элемента указанного элемента, либо -1, если таковой не найден
- Методы FindIndex, FindLastIndex и IndexOf имеют по две перегрузки с сигнатурами (int, Predicate<T>) и (int, int, Predicate<T>), позволяющие производить поиск в части списка, начиная с заданного индекса и с заданной длиной

МЕТОДЫ СОРТИРОВКИ

- Класс `List<T>` позволяет сортировать свои элементы с помощью метода `Sort ()`, в котором реализован алгоритм быстрой сортировки
- Для использования доступно несколько перегрузок этого метода:
 - public void** `List<T>.Sort () ;`
 - public void** `List<T>.Sort(Comparison<T>);`
 - public void** `List<T>.Sort(IComparer<T>);`
 - public void** `List<T>.Sort(Int32, Int32, IComparer<T>);`
- Использовать метод `Sort ()` без аргументов можно только в том случае, когда элементы коллекции реализуют интерфейс `IComparable`

ИНТЕРФЕЙС ICOMPARABLE

- Необобщенный интерфейс `IComparable` объявляет единственный метод `CompareTo(object)`, который указывает, находится ли текущий экземпляр в порядке сортировки перед, после или на той же позиции, что и объект, указанный в качестве аргумента метода `CompareTo`

ИНТЕРФЕЙС ICOMPARABLE

- Реализация метода CompareTo должна возвращать значение типа Int32, которое имеет значение:
 - ✓ больше 0, если текущий экземпляр следует в порядке сортировки за объектом-аргументом;
 - ✓ равное нулю, если они находятся в одной позиции сортировки;
 - ✓ меньше 0, если текущий экземпляр предшествует объекту-аргументу в порядке сортировки
- Все числовые типы, а также типы String, Char и DateTime реализуют интерфейс IComparable

МЕТОД SORT(ICOMPARER<T>)

- Если сортировка должна быть выполнена способом, отличным от поддерживаемого по умолчанию типом элементов, то потребуются передавать объект, реализующий обобщенный интерфейс IComparer<T>
- Для этого метода существует перегрузка, позволяющая сортировать часть массива

ИНТЕРФЕЙС ICOMPARER<T>

□ Обобщенный интерфейс IComparer<T> объявляет единственный метод

`int Compare(T x, T y)`

□ Этот метод часто называют *компаратором*; возвращаемые им значения определяются по тем же правилам, что и для метода CompareTo:

- ✓ больше нуля, если параметр x больше параметра y;
- ✓ равно нулю, если параметр x равен параметру y;
- ✓ меньше нуля, если параметр x меньше параметра y

МЕТОД SORT(COMPARISON<T>)

- Другой способ сортировки состоит в применении перегруженного метода сортировки с параметром-делегатом `Comparison<T>`

ДЕЛЕГАТ COMPARISON<T>

□ Comparison<T> представляет собой делегат метода, принимающего два параметра типа T и возвращающего тип int

```
public delegate int Comparison<T>( T x, T y )
```

□ Если значения параметров эквиваленты, метод должен вернуть 0

□ Если первый параметр меньше второго, должно быть возвращено значение меньше нуля; в противном случае возвращается значение больше нуля

ДЕЛЕГАТ COMPARISON<T>

- При вызове метода Sort в качестве аргумента может быть указано имя любой функции, соответствующей объявлению этого делегата
- Например
- В этом примере для сравнения строк используется пользовательский компаратор, который производит лексикографическое сравнение строк только в случае равенства их длин

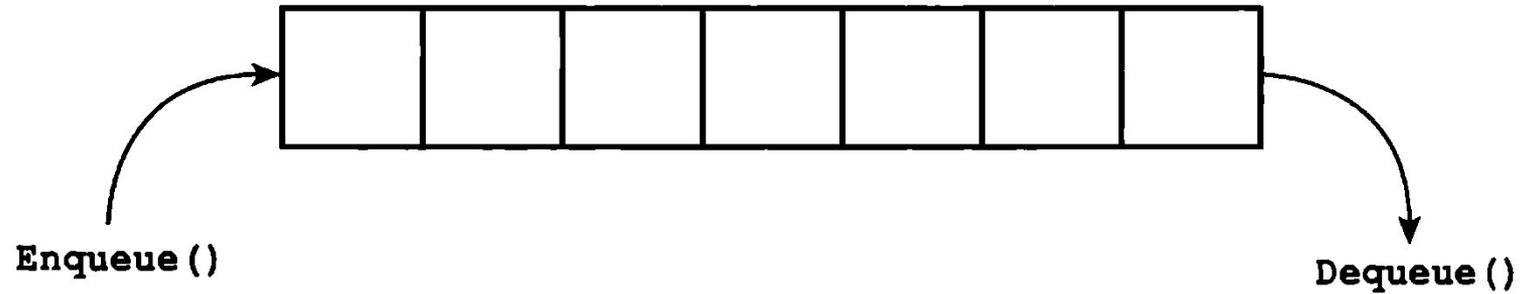
ОЧЕРЕДИ

- Очередь (queue) — это коллекция, в которой элементы обрабатываются по схеме "первый вошел, первый вышел" (first in, first out — FIFO)
- Элемент, вставленный в очередь первым, первым же и удаляется из очереди
- Очередь реализуется с помощью класса `Queue<T>` из пространства имен `System.Collections.Generic`
- Внутри класс `Queue<T>` использует массив типа `T`, подобно тому, как это делает класс `List<T>`
- Класс `Queue<T>` реализует интерфейсы `IEnumerable<T>` и `ICollection`, но не `ICollection<T>`

ОЧЕРЕДИ

- Интерфейс `ICollection<T>` не реализован, поскольку он определяет методы `Add ()` и `Remove ()`, которые не должны быть доступны для очереди
- Класс `Queue<T>` не реализует интерфейс `ICollection<T>`, поэтому обращаться к элементам очереди через индексатор нельзя.
- Очередь позволяет лишь добавлять элементы, при этом элемент помещается в конец очереди (методом `Enqueue ()`), а также получать элементы из головы очереди (методом `Dequeue()`)

ПРОСТАЯ ОЧЕРЕДЬ



СОЗДАНИЕ ОЧЕРЕДИ

- Подобно `List<T>` класс `Queue<T>` имеет три конструктора:
 - ✓ `public Queue<T>()` – конструктор по умолчанию, создающий пустой список с нулевой емкостью;
 - ✓ `public Queue<T>(Int32)` – конструктор, создающий список с заданной начальной емкостью;
 - ✓ `public Queue<T>(IEnumerable<T>)` – конструктор, создающий список и копирующий в нее элементы указанной коллекции
- Конструктор по умолчанию создает пустую очередь емкости 4

СОЗДАНИЕ ОЧЕРЕДИ

- По мере добавления элементов емкость очереди при необходимости удваивается
- Отметим, что конструктор по умолчанию необобщенного класса `Queue` отличается тем, что создает начальный массив из 32 пустых элементов.

МЕТОДЫ КЛАССА QUEUE<T>

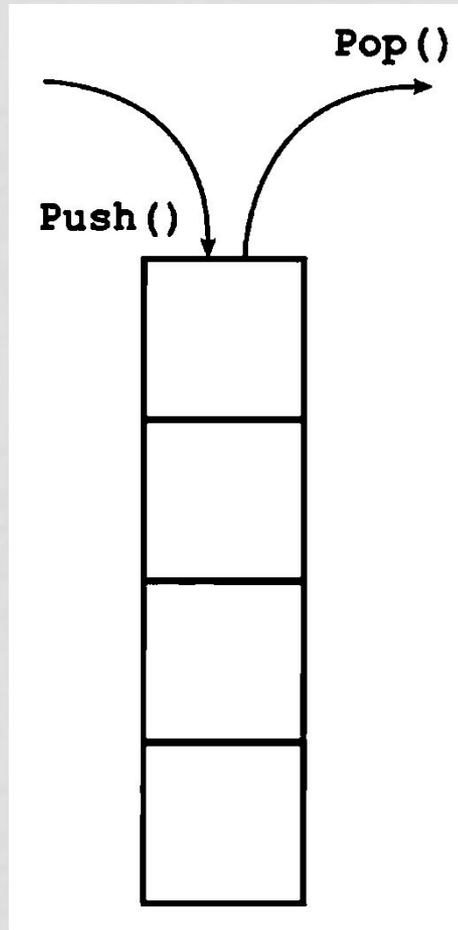
Название метода	Назначение метода
Clear()	Удаляет все элементы из очереди
Contains(T)	Определяет, содержится ли указанное значение в очереди
Enqueue(T)	Вставляет элемент в конец очереди
T Dequeue()	Удаляет элемент из начала очереди
T Peek()	Читает элемент из начала очереди, но не удаляет его
TreatExcess()	Устанавливает емкость равной фактическому количеству элементов в очереди, если это количество составляет менее 90% текущей емкости

- Класс Queue<T> содержит свойство для чтения Count(), возвращающее количество элементов в очереди

СТЕКИ

- Стек (stack) — это коллекция, работающая по принципу "последний вошел, первый вышел" (last in, first out — LIFO)
- Элемент, добавленный к стеку последним, читается первым
- Подобно классу `Queue<T>`, класс `Stack<T>` реализует интерфейсы `IEnumerable<T>` и `ICollection`

ПРОСТОЙ СТЕК



МЕТОДЫ КЛАССА STACK<T>

Название метода	Назначение метода
Clear()	Удаляет все элементы из стека
Contains(T)	Определяет, содержится ли указанное значение в стеке
Push(T)	Вставляет элемент в вершину стека
T Pop()	Удаляет элемент из вершины стека
T Peek()	Читает элемент из вершины стека, но не удаляет его
TreatExcess()	Устанавливает емкость равной фактическому количеству элементов в стеке, если это количество составляет менее 90% текущей емкости

- Класс Stack<T> содержит свойство для чтения Count(), возвращающее количество элементов в стеке

МНОЖЕСТВА

- Коллекция, содержащая только отличающиеся элементы, называется множеством (set)
- Множества реализуются обобщенным классом `HashSet<T>`, реализующим интерфейс `ISet<T>`, а также интерфейсы `ICollection<T>`, `IEnumerable<T>`, `IEnumerable`
- Интерфейс `ISet<T>` предоставляет методы для создания объединения нескольких множеств, пересечения множеств и определения, является ли одно множество надмножеством или подмножеством другого

КОНСТРУКТОРЫ МНОЖЕСТВ

- Класс коллекции `HashSet<T>` имеет несколько перегрузок конструктора, в том числе:
 - ✓ `public HashSet<T>()` – конструктор по умолчанию, создающий пустое множество и использующий компаратор по умолчанию для сравнения элементов множества;
 - ✓ `public HashSet<T>(IEnumerable<T>)` – конструктор, создающий множество с элементами, скопированными из указанной коллекции, и использующий компаратор по умолчанию для сравнения элементов множества;

КОНСТРУКТОРЫ МНОЖЕСТВ

- ✓ `public HashSet<T>(IEqualityComparer<T>)` – конструктор создающий пустое множество и использующий указанный компаратор для сравнения элементов множества;
 - ✓ `public HashSet<T>(IEnumerable<T>, IEqualityComparer<T>)` – конструктор, создающий множество с элементами, скопированными из указанной коллекции, и использующий указанный компаратор для сравнения элементов множества;
- Коллекция `HashSet <T>` не сортируется и не может содержать повторяющихся элементов
- Класс `HashSet <T>` содержит свойство для чтения `Count()`, возвращающее количество элементов в множестве

МЕТОДЫ КЛАССА HASHSET<T>

Название метода	Назначение метода
Add(T)	Добавление элемента в множество
Remove(T)	Удаление указанного элемента из множества
RemoveWhere (Predicate<T>)	Удаление из множества всех элементов, удовлетворяющих заданному условию
Clear()	Удаление из множества всех элементов
Contains(T)	Проверка вхождения элемента в множество
UnionWith (IEnumerable<T>)	Объединение множества с указанной коллекцией
IntersectWith (IEnumerable<T>)	Пересечение множества с указанной коллекцией
ExceptWith (IEnumerable<T>)	Удаляет все элементы указанной коллекции из текущего множества (вычитание множеств)

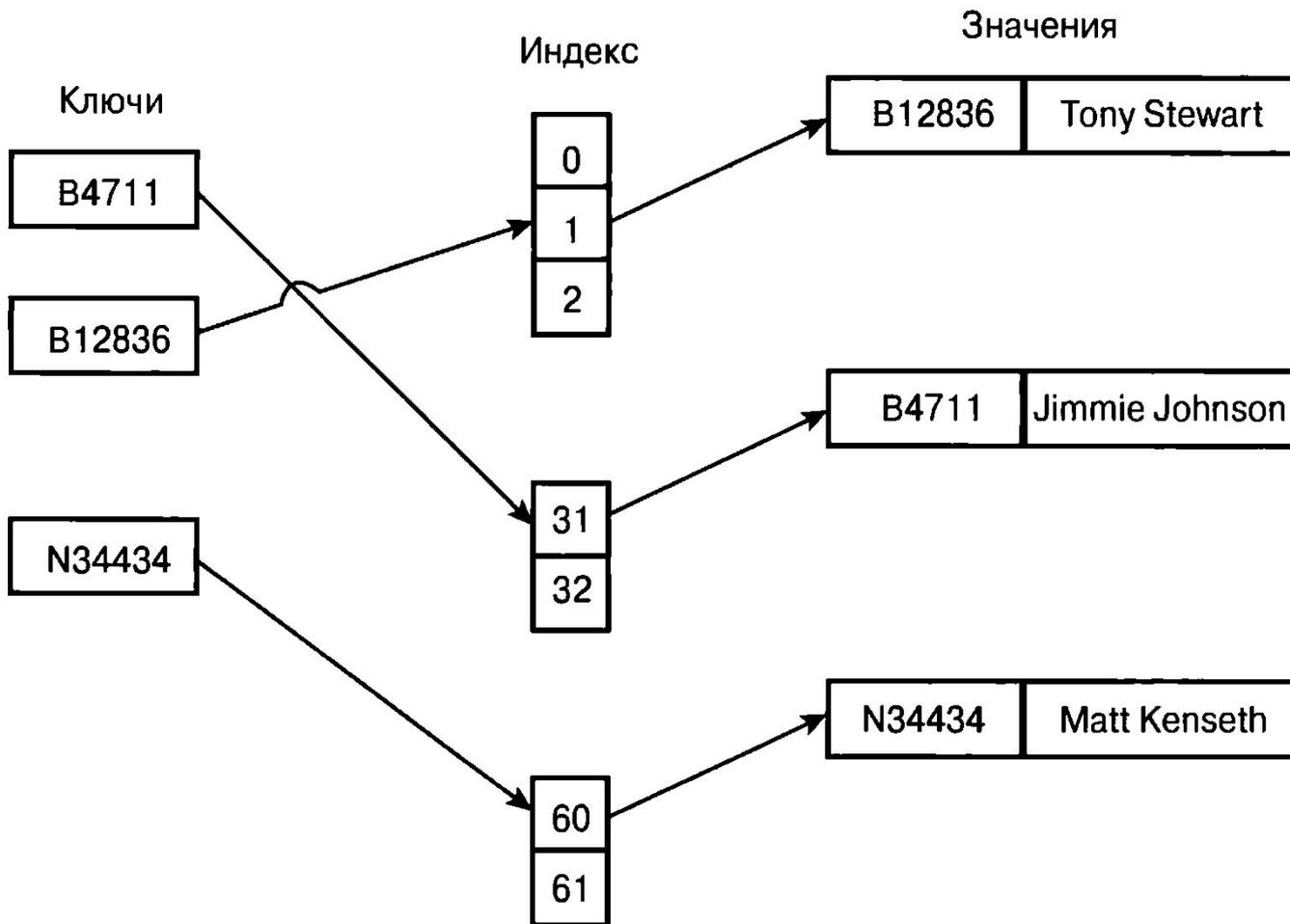
СЛОВАРИ

- Словарь (dictionary) представляет собой сложную структуру данных, позволяющую обеспечить доступ к элементам по ключу
- Под элементом ключа здесь и далее понимается пара «ключ-значение»
- Главное свойство словарей – быстрый поиск на основе ключей

СЛОВАРИ

- В основе такого поиска лежит использование механизма *индексации* – сопоставления каждому элементу целого значения, называемого *индексом*
- Механизм индексации используется в хорошо известных массивах

ПРИМЕР СЛОВАРЯ



ХЕШ-ФУНКЦИИ

- Индексация в словарях отличается от индексации в массивах тем, что индекс элемента является целочисленной функцией ключа
- Такая функция называется хеш-функцией, а возвращаемый ею индекс – хеш-кодом
- В идеале хеш-функция должна обеспечивать взаимно однозначное соответствие между множеством ключей и множеством индексов
- Однако на практике такую функцию построить невозможно

КОЛЛИЗИИ

- При всех известных способах построения хеш-функций возникают ситуации, называемые *коллизиями*, когда для двух различных значений ключа получается одно и то же значение индекса
- В таких случаях используются различные алгоритмы *разрешения коллизий*, например, замена уже занятого индекса ближайшим к нему незанятым индексом
- Еще одним требованием к хеш-функции, обусловленным частотой ее использования, является высокая скорость вычисления индекса

ТИП КЛЮЧА

- Тип, используемый в качестве ключа словаря, должен переопределять метод `GetHashCode ()` класса `Object`
- Всякий раз, когда класс словаря должен найти местоположение элемента, он вызывает метод `GetHashCode ()`
- Помимо реализации `GetHashCode ()` тип ключа также должен реализовывать единственный метод интерфейса `IEquatable<T> Equals ()` либо переопределять метод `Equals ()` класса `Object`

ТИП КЛЮЧА

- Поскольку разные ключи могут возвращать один и тот же хеш-код, метод `Equals ()` используется при сравнении ключей словаря
- Словарь проверяет два ключа `A` и `B` на эквивалентность, вызывая `A.Equals (B)`
- Это означает, что требуется обеспечить истинность следующего утверждения:
 - ✓ Если истинно `A.Equals (B)`, значит, `A.GetHashCode ()` и `B.GetHashCode ()` всегда должны возвращать один и тот же хеш-код

КОНСТРУКТОРЫ СЛОВАРЯ

- Класс коллекции Dictionary<T> имеет несколько перегрузок конструктора, в том числе:
 - ✓ `public Dictionary<TKey, TValue> ()` – конструктор по умолчанию, создающий пустое словарь и использующий компаратор по умолчанию для сравнения ключей;
 - ✓ `public Dictionary<TKey, TValue> (IDictionary<TKey, TValue>)` – конструктор, создающий словарь с элементами, скопированными из указанного словаря, и использующий компаратор по умолчанию для сравнения ключей;

КОНСТРУКТОРЫ СЛОВАРЯ

- ✓ `public Dictionary<TKey, TValue>(IEqualityComparer<TKey>)` – конструктор создающий пустой словарь и использующий указанный компаратор для сравнения ключей;
- ✓ `public Dictionary<TKey, TValue>(Int32)` – конструктор, создающий пустой словарь с заданной емкостью;

МЕТОДЫ КЛАССА DICTIONARY<TKEY, TVALUE>

Название метода	Назначение метода
Add(TKey, TValue)	Добавление элемента в словарь
Remove(TKey)	Удаление из словаря элемента с заданным ключом
Clear()	Удаление из словаря всех элементов
ContainsKey(TKey)	Проверка вхождения в словарь элемента с заданным ключом
ContainsValue(TValue)	Проверка вхождения в словарь элемента с заданным значением
GetHashCode()	Возвращает хэш-код для текущего элемента
ToString()	Возвращает строковое представление текущего элемента
TryGetValue(TKey, out TValue)	Значение true , если словарь содержит элемент с указанным ключом; в противном случае — значение false

СВОЙСТВА КЛАССА DICTIONARY<TKEY, TValue>

Название свойства	Назначение свойства
Count	Количество элементов в словаре (только для чтения)
Item	Ключ текущего элемента
Keys	Коллекция всех ключей в словаре
Values	Коллекция всех значений в словаре