

Лабораторная работа № 16

Тема: Разработка программ с использованием обобщений и коллекций

Цель работы: Формирование умений и навыков использования обобщений и коллекций в частности Hashtable.

Hashtable — это набор пар ключ / значение, которые упорядочены на основе хеш-кода ключа. Или, другими словами, Hashtable используется для создания коллекции, которая использует хеш-таблицу для хранения. Как правило, он оптимизировал поиск, вычисляя хеш-код каждого ключа и автоматически сохраняя его в другой корзине, и при обращении к значению из хеш-таблицы в это время он сопоставляет хеш-код с указанным ключом. Это неуниверсальный тип коллекции, который определен в пространстве имен *System.Collections*.

Важные моменты:

- В Hashtable ключ не может быть нулевым, но значение может быть.
- В Hashtable ключевые объекты должны быть неизменяемыми, если они используются в качестве ключей в Hashtable.
- Емкость Hashtable — это количество элементов, которое может содержать Hashtable.
- Хэш-функция предоставляется каждым ключевым объектом в Hashtable.
- Хэш реализует КЛАСС *IDictionary*, *ICollection*, *IEnumerable*, *ISerializable*, *IDeserializationCallback* и *ICloneable* интерфейсов.
- В хеш-таблице вы можете хранить элементы одного типа и разных типов.
- Элементы хеш-таблицы, которая является парой ключ / значение, хранятся в *DictionaryEntry*, поэтому вы также можете преобразовать пары ключ / значение в *DictionaryEntry*.
- В Hashtable ключ должен быть уникальным. Дублирование ключей не допускается.

Как создать Hashtable?

Класс Hashtable предоставляет 16 различных типов конструкторов, которые используются для создания хеш-таблицы, здесь мы используем только конструктор *Hashtable()*. Чтобы узнать больше о конструкторах Hashtable, вы можете обратиться к [С # | Класс Hashtable](#). Этот конструктор используется для создания экземпляра класса Hashtable, который является пустым и имеет начальную емкость по умолчанию, коэффициент загрузки, поставщик хэш-кода и средство сравнения. Теперь давайте посмотрим, как создать хеш-таблицу с помощью конструктора Hashtable():

Шаг 1: Включить System.Collections имен в программе с помощью использования ключевого слова:

```
using System.Collections;
```

Шаг 2: Создайте хеш-таблицу, используя класс Hashtable, как показано ниже:

```
Hashtable hashtable_name = new Hashtable();
```

Шаг 3: Если вы хотите добавить пару ключ / значение в вашу хеш-таблицу, используйте метод `Add ()` для добавления элементов в вашу хеш-таблицу. Также вы можете сохранить пару ключ / значение в вашей хеш-таблице без использования метода [Add \(\)](#).

Пример:

```
// C # программа для иллюстрации того, как
// создать хеш-таблицу
using System;
using System.Collections;
class GFG {
    // Основной метод
    static public void Main()
    {
        // Создать хеш-таблицу
        // Использование класса Hashtable
        Hashtable my_hashtable1 = new Hashtable();
        // Добавляем пару ключ / значение
        // в хеш-таблице
        // Использование метода Add ()
        my_hashtable1.Add("A1", "Welcome");
        my_hashtable1.Add("A2", "to");
        my_hashtable1.Add("A3", "GeeksforGeeks");
        Console.WriteLine("Key and Value pairs from my_hashtable1:")
        foreach(DictionaryEntry ele1 in my_hashtable1)
        {
            Console.WriteLine("{0} and {1} ", ele1.Key, ele1.Value);
        }

        // Создать другую хеш-таблицу
        // Использование класса Hashtable
        // и добавляем пары ключ / значение
        // без использования метода Add
        Hashtable my_hashtable2 = new Hashtable() {
            {1, "hello"},
            {2, 234},
            {3, 230.45},
            {4, null}};
```

```

        Console.WriteLine("Key and Value pairs from my_hashtable2:")
        foreach(var ele2 in my_hashtable2.Keys)
        {
            Console.WriteLine("{0}and {1}", ele2,
                               my_hashtable2[ele2]);
        }
    }
}

```

Вывод:

```

Key and Value pairs from my_hashtable1:
A3 and GeeksforGeeks
A2 and to
A1 and Welcome
Key and Value pairs from my_hashtable2:
4and
3and 230.45
2and 234
1and hello

```

Как удалить элементы из хеш-таблицы?

В Hashtable вам разрешено удалять элементы из хеш-таблицы. Класс Hashtable предоставляет два разных метода для удаления элементов, и методы:

- **Очистить** : этот метод используется для удаления всех объектов из хеш-таблицы.
- **Remove** : этот метод используется для удаления элемента с указанным ключом из хеш-таблицы.

Пример:

```

// C # программа для иллюстрации того, как
// удаляем элементы из хеш-таблицы
using System;
using System.Collections;
class GFG {

    // Основной метод
    static public void Main()
    {
        // Создать хеш-таблицу
        // Использование класса Hashtable
        Hashtable my_hashtable = new Hashtable();

        // Добавляем пару ключ / значение
        // в хеш-таблице

```

```

// Использование метода Add ()
my_hashtable.Add("A1", "Welcome");
my_hashtable.Add("A2", "to");
my_hashtable.Add("A3", "GeeksforGeeks");

// Используем метод удаления
// удаляем пару ключ / значение A2
my_hashtable.Remove("A2");
Console.WriteLine("Key and Value pairs :");
foreach(DictionaryEntry ele1 in my_hashtable)
{
    Console.WriteLine("{0} and {1} ", ele1.Key, ele1.Value);
}

// Перед использованием метода Clear
Console.WriteLine("Total number of elements present"+
    " in my_hashtable:{0}", my_hashtable.Count);

my_hashtable.Clear();

// После использования метода Clear
Console.WriteLine("Total number of elements present in"+
    " my_hashtable:{0}", my_hashtable.Count);
}
}

```

Вывод:

```

Key and Value pairs :
A3 and GeeksforGeeks
A1 and Welcome
Total number of elements present in my_hashtable:2
Total number of elements present in my_hashtable:0

```

Как проверить наличие пары ключ / значение в хеш-таблице?

В хеш-таблице вы можете проверить, присутствует ли данная пара или нет, используя следующие методы:

- **Содержит** : Этот метод используется для проверки того, содержит ли Hashtable определенный ключ.
- **ContainsKey** : этот метод также используется для проверки того, содержит ли Hashtable определенный ключ.
- **ContainsValue** : этот метод используется для проверки того, содержит ли Hashtable определенное значение.

Пример:

```
// C # программа для иллюстрации того, как
// проверить наличие ключа / значения
// в хеш-таблице или нет

using System;
using System.Collections;

class GFG {

    // Основной метод
    static public void Main()
    {

        // Создать хеш-таблицу
        // Использование класса Hashtable
        Hashtable my_hashtable = new Hashtable();

        // Добавление пары ключ / значение в хеш-таблицу
        // Использование метода Add ()
        my_hashtable.Add("A1", "Welcome");
        my_hashtable.Add("A2", "to");
        my_hashtable.Add("A3", "GeeksforGeeks");

        // Определяем, является ли данный
        // ключ присутствует или нет
        // используя метод Contains
        Console.WriteLine(my_hashtable.Contains("A3"));
        Console.WriteLine(my_hashtable.Contains(12));
        Console.WriteLine();

        // Определяем, является ли данный
        // ключ присутствует или нет
        // используя метод ContainsKey
        Console.WriteLine(my_hashtable.ContainsKey("A1"));
        Console.WriteLine(my_hashtable.ContainsKey(1));
        Console.WriteLine();

        // Определяем, является ли данный
        // значение присутствует или нет
```

```

        // используя метод ContainsValue
        Console.WriteLine(my_hashtable.ContainsValue("geeks"));
        Console.WriteLine(my_hashtable.ContainsValue("to"));
    }
}

```

Вывод:

```

True
False

True
False

False
True

```

Практическая часть

Задание 1.

Реализовать класс. Для реализации использовать либо Dictionary либо HashSet. Конструкторы из файла, случайно n элементов от 1 до n, через консоль руками. Перегрузка ToString для корректного вывода. Класс должен быть **ОБОБЩЕННЫМ**, но гарантировать работу с типами данных int, double, string

Вариант	Задание
1	Класс-контейнер МНОЖЕСТВО (обобщенный). Реализовать операции: [] — доступа по индексу; * — пересечение множеств; -- — удаление случайного по порядку элемента из множества.
2	Класс-контейнер МНОЖЕСТВО (обобщенный). ==, != — проверка на (не)равенство; число — принадлежность числа множеству; - значение — удаление значения из множества.
3	Класс-контейнер МНОЖЕСТВО (обобщенный). Реализовать операции: [] — доступа по индексу; .Length — определение размера множества; +число — добавляет константу ко всем элементам множества; -n — удаляет n элементов из конца множества
4	Класс-контейнер МНОЖЕСТВО (обобщенный). -n — удаляет n элементов из начала множества; +n — добавляет элемент в множество. >> число — сдвигает каждое значение множества в байткоде на число бит влево. << число — сдвигает каждое значение множества в байткоде на число бит вправо.
5	Класс-контейнер МНОЖЕСТВО (обобщенный). Реализовать операции: [] — доступа по индексу; +число — добавляет число случайных элементов, сгенерированных при помощи random. .Length — определение размера множества.

6	Класс-контейнер МНОЖЕСТВО (обобщенный). -(минус) — удаляет элемент из множества (постфиксная операция удалят элемент из конца, префиксная из начала), -- — удаление случайного по порядку элемента из множества
7	Класс-контейнер МНОЖЕСТВО (обобщенный). Реализовать операции:[] — доступа по индексу; * множество — возвращает объединение множеств. +число — к каждому элементу добавляется число.
8	Класс-контейнер МНОЖЕСТВО (обобщенный). + множество — объединение множеств; ++ — добавление случайного элемента в множество.
9	Класс-контейнер МНОЖЕСТВО (обобщенный). Реализовать операции:[] — доступа по индексу; ==, != — проверка на (не)равенство; &значение – принадлежность значения множеству
10	Класс-контейнер МНОЖЕСТВО (обобщенный). Реализовать операции:[] — доступа по индексу; -значение — уменьшение всех элементов на значение элемента. *— пересечение множеств;