**Группа** ИС 3-1

**ФИО:** Мозыркий А.Д.

**Курс:** C# Professional

**Урок 2:** Системные коллекции

**Теория**

• Наборы — классы, предназначенные для группировки связанных объектов, управления ими и обработки их в циклах, — являются одним из основных инструментов программиста.

• В .NET Framework в пространстве имен System.Collections реализовано большое количество специализированных наборов, которые значительно упрощают и ускоряют процесс разработки.

• ArrayList - простой, поддерживающий индексирование и изменение размера набор объектов. Является простым неупорядоченным контейнером для объектов любого типа. В наборах разрешается хранить значимые типы, но сначала их необходимо преобразовать к ссылочному типу, выполнив операцию упаковки (boxing). Метод AddRange позволяет добавлять диапазоны (или группу элементов) любых объектов, поддерживающих интерфейс ICollection (включая массивы, объекты ArrayList и большинство других наборов). Методы Add и AddRange добавляют элементы в конец набора. Поскольку наборы ArrayList являются динамическими, они поддерживают вставку объектов в заданном положении. Для этого ArrayList предоставляет методы Insert и InsertRange. Интерфейс IEnumerable унифицирует перебор элементов набора в цикле. .NET Framework также поддерживает общий интерфейс — образец API наборов. Этот интерфейс называется ICollection и происходит от интерфейса IEnumerable. Это означает, что любой набор, поддерживающий интерфейс ICollection, также поддерживает интерфейс IEnumerable. Интерфейс ICollection содержит в себе 3 свойства(Count, IsSynchronized, SyncRoot) и один метод (CopyTo).

• .NET Framework поддерживает еще один интерфейс для доступа к элементам списка: IList. Этот интерфейс наследуется непосредственно от ICollection. Если класс поддерживает интерфейс IList, он также поддерживает интерфейсы ICollection и IEnumerable.

• Последовательные списки рекомендуется использовать в том случае, когда необходимо обрабатывать списки объектов последовательно, а не в произвольном порядке. Последовательные списки представлены двумя классами (Queue и Stack).

• Queue - набор объектов, организованный по принципу «первым вошел, первым вышел» (FIFO; first-in, first-out). В отличии от ArrayList, в котором обращение к элементу и удаление его из набора представлены различными операциями, Queue объединяет эти операции в комбинированном методе Dequeue. Эти операции исходно логически связаны в силу специфики класса Queue. Создав экземпляр класса, можно вызывать метод Enqueue для добавления элементов в очередь и метод Dequeue для удаления элементов из нее. Класс Queue позволяет добавлять в список дублирующиеся элементы и null-значения, поэтому стандартные методы Dequeue и Реек не помогут определить, пуста ли очередь Queue.

• Stack - набор объектов, организованный по принципу «последним вошел, первым вышел» (LIFO; last-in, first-out). Работу данного класса можно сравнить с колодой карт из которой карты можно убрать и ложить обратно только сверху. Работа с классом Stack похожа на работу с классом Queue: создают экземпляр класса Stack, затем для добавления элементов в стек вызывают метод Push, а для удаления — метод Pop. Оба класса Stack и Queue поддерживают метод Реек для просмотра следующего элемента в наборе без его удаления.

• Hashtable - набор пар объектов «имя-значение», предоставляющих доступ к элементам как по имени, так и по индексу в наборе. Чтобы получить доступ к данным, уже добавленным в словарь, достаточно вызвать индексатор с требуемым ключом. Если из словаря требуется вывести значения, следует указать итератору, что он работает с объектами типа

DictionaryEntry. Класс Hashtable требует уникальности хэш-кодов, а не связанных с ними значений. Если же попытаться сохранить разные значения с одним и тем же ключом, то второе значение заменит первое. Класс Hashtable чувствителен к регистру. Существенным недостатком данного класса является высокое потребление ресурсов, при использовании маленьких наборов (меньше десяти элементов) это снижает производительность.

• Интерфейс IEqualityComparer предоставляет возможность реализовать настраиваемое сравнение коллекций в отношении равенства. Это означает, что можно создать свое собственное определение равенства и указать, что это определение должно использоваться для типа коллекции, которая принимает интерфейс IEqualityComparer. В .NET Framework конструкторы типов коллекций Hashtable, NameValueCollection и OrderedDictionary принимают этот интерфейс. Данный интерфейс поддерживает только сравнения в отношении равенства. Настройка сравнения для сортировки или упорядочения реализуется с помощью интерфейса IComparer.

• SortedList - упорядоченный набор пар объектов «имя-значение». Чтобы упорядочить коллекцию, представленную SortedList вызов метода Sort не требуется. SortedList сам упорядочит элементы в момент добавления

• ListDictionary - набор, подходящий для хранения небольших списков объектов. Он оптимален для небольших наборов (до 10-ти элементов), поскольку устроен как простой массив.

• HybridDictionary - набор, в котором элементы хранятся в ListDictionary, если их мало, либо в Hashtable, если их много. Используется в тех случаях, когда определить размер коллекции изначально невозможно.

• BitArray - компактный набор значений типа Boolean. Класс BitArray поддерживает нединамическое изменение размера. Размер набора необходимо указывать при создании экземпляра класса BitArray. После этого для изменения размера можно воспользоваться свойством Length. Элементы набора BitArray принимают одно из двух значений: true или false, поэтому сама концепция добавления или удаления элементов к такому набору неприменима.

• Структура BitVector32 очень удобна для управления отдельными битами больших чисел. BitVector32 хранит свои данные как 32-разрядное целое число. Все операции над объектами BitVector32 в действительности изменяют значение целого внутри структуры.

Структура BitVector32 позволяет последовательно создавать битовые маски, вызывая статический метод CreateMask. Эти маски можно использовать вместе с индексатором структуры BitVector32 для установки или получения значения отдельного бита, соответствующего маске. Также коллекция BitVector32 очень удобна для работы с отдельными битами и поддерживает упаковку битов. Упаковка часто позволяет сэкономить память, необходимую для хранения маленьких чисел.

• StringCollection - простой, поддерживающий изменение размера, набор строк. Код, добавляющий строки в набор, выглядит так же, как и в приведенном выше примере с использованием ArrayList. Единственное отличие в том, что попытка добавления объекта с типом, отличным от строкового, влечет ошибку компиляции. К тому же теперь, получая строки из набора, вы работаете не с объектами, а именно со строками, что устраняет необходимость приведения типов.

• StringDictionary - набор пар строк «имя-значение», предоставляющих доступ к элементам как по имени, так и по индексу в наборе. Он работает как Hashtable, но и ключ, и значение

должны быть строками. Важно понимать, что по умолчанию ключи для объектов StringDictionary нечувствительны к регистру, поэтому ключи "Fourth" и "FOURTH" эквиваленты.

• NameValueCollection - набор пар строк «имя-значение», предоставляющих доступ к элементам как по имени, так и по индексу в наборе. В классе NameValueCollection допускается несколько значений, соответствующих одному ключу, к тому же, значения можно получать не только по ключу, но и по индексу. Таким образом, при работе с классом NameValueCollection можно хранить несколько значений с одним ключом. Метод Add позволяет делать это.

• Класс CollectionUtil поддерживает создание объектов Hashtable и SortedList, нечувствительных к регистру. Для создания этих объектов достаточно вызвать метод CreateCaseInsensitiveHashtable или CreateCaselnsensitiveSortedList.

• Обобщения — это типы, которые принимают имена других типов в качестве параметров, и используют их. Вместо того, чтобы создавать строго типизированные наборы для каждого типа, достаточно создать набор, которые можно настроить на работу с любым типом.

• Класс-обобщение LinkedList имеет одну особенность — реализацию перечислителя ILinkedListEnumerator, что позволяет перечислять значения списка без использования объектов LinkedListNode.

**Практика**

**Задание 1**

Рис.1.1 Реализация класса

Рис.1.2 Реализация коллекции

Рис.1.3 Реализация кода программы

Рис.1.4 Результат работы программы

**Задание 2**

Рис.2.1 Реализация класса

Рис.2.2 Реализация коллекции

Рис.2.3 Реализация кода программы

Рис.2.4 Результат работы программы

**Задание 3**

Рис.3.1 Реализация класса

Рис.3.2 Реализация коллекции

Рис.3.3 Реализация кода программы

Рис.3.4 Результат работы программы