Лабораторная работа №5

Коллекции

Цель работы: изучить работу с коллекциями в языке C#.

Теоретическая часть

В C# коллекция представляет собой совокупность объектов.

В среде .NET Framework имеется немало интерфейсов и классов, в которых определяются и реализуются различные типы коллекций.

Коллекции упрощают решение многих задач программирования благодаря тому, что предлагают готовые решения для создания целого ряда типичных, но порой трудоемких для разработки структур данных. Главное преимущество коллекций заключается в том, что они стандартизируют обработку групп объектов в программе.

Все коллекции разработаны на основе набора четко определенных интерфейсов.

Типы коллекций:

1) Необобщенные

   Классы и интерфейсы необобщенных коллекций находятся в пространстве имен `System.Collections`. Они реализуют ряд основных структур данных, включая динамический массив, стек, очередь, а также словари. В отношении необобщенных коллекций важно иметь в виду следующее: они оперируют данными типа `object`. Необобщенные коллекции могут служить для хранения данных любого типа, причем в одной коллекции допускается наличие разнотипных данных.

2) С поразрядной организацией

Объявляется в пространстве имен `System.Collections`. Коллекция типа `BitArray` поддерживает поразрядные операции, т.е. операции над отдельными двоичными разрядами, например И, ИЛИ, исключающее ИЛИ, а следовательно, она существенно отличается своими возможностями от остальных типов коллекций.

3) Специальные

   Объявляются в `System.Collections.Specialized`. Специальные коллекции оперируют данными конкретного типа или же делают это каким-то особым образом. Например, имеются специальные коллекции для символьных строк, а также специальные коллекции, в которых используется однонаправленный список.

4) Обобщенные

   Объявляются в пространстве имен `System.Collections.Generic`. Обобщенные коллекции обеспечивают обобщенную реализацию нескольких стандартных структур данных, включая связные списки, стеки, очереди и словари. Такие коллекции являются типизированными в силу их обобщенного характера. Это означает, что в обобщенной коллекции могут храниться только такие элементы данных, которые совместимы по типу с данной коллекцией. Благодаря этому исключается случайное несовпадение типов.

5) Параллельные

   Это обобщенные коллекции, определенные в `System.Collections.Concurrent`. Поддерживают многопоточный доступ к коллекции.

Основополагающим для всех коллекций является понятие перечислителя.

Перечислитель обеспечивает стандартный способ поочередного доступа к элементам коллекции. Следовательно, он перечисляет содержимое коллекции.

В каждой коллекции должна быть реализована обобщенная или необобщенная форма интерфейса `IEnumerable`, поэтому элементы любого класса коллекции должны быть доступны посредством методов, определенных в интерфейсе `IEnumerator` или `IEnumerator<T>`.

С перечислителем непосредственно связано другое средство - итератор.

Это средство упрощает процесс создания классов коллекций, например специальных, поочередное обращение к которым организуется в цикле `foreach`. Итератор представляет собой блок кода, который использует оператор `yield` для перебора набора значений. Данный блок кода может представлять тело метода, оператора или блок `get` в свойствах.

Итератор использует две специальных инструкции:

- `yield return` определяет возвращаемый элемент
- `yield break` указывает, что последовательность больше не имеет элементов

Практическая часть

Примечание: нельзя использовать методы класса `Array`.

Разработать класс `DynamicArray<T>`. Класс должен содержать:

1) Приватное поле для хранения элементов динамического массива `T[ ]`
2) Свойство `Count` – количество элементов в массиве
3) Свойство `Capacity` – емкость массива
4) Конструктор без параметров. Значение `Capacity` задается по умолчанию (например, 20)

5) Конструктор, принимающий значение `Capacity`

6) Метод `Add(T element)` для вставки элемента в конец массива

7) Метод `Add(IEnumerable<T> elements)` для добавления коллекции в конец массива

8) Метод `Insert(T element, int position)` для вставки элемента по указанной позиции

9) Метод `RemoveAt(int position)` для удаления элемента по указанной позиции

10) Метод `IncreaseCapacity(int n)` для увеличения емкости массива на n элементов

11) Метод `IEnumerator<T> GetEnumerator()` (реализовать интерфейс `IEnumerable`). Требуется самостоятельно реализовать интерфейс `IEnumerator<T>` (свойство `Current`, методы `MoveNext` и `Reset`) в отдельном классе

Содержание отчета

1. Титульный лист
2. Цель работы
3. Задание
4. Код программы
5. Результат выполнения