ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР
ПРОГРАММИРОВАНИЯ И ВЫСОКИХ ТЕХНОЛОГИЙ

2020г.

Коллекции

- ✓ Стандартные коллекции
- ✓ Коллекции-списки
- ✓ Коллекции-словари
- ✓ Специальные типы коллекций: стек, очередь, хэш-таблица

Стандартные коллекции

Пространства имен содержащие коллекции:

System.Collections – содержит типы, в которых элемент коллекции представлен как **object** (слаботипизированные коллекции).

System.Collections.Generic – универсальные классы и интерфейсы коллекций.

System.Collections.Specialized – специальные классы коллекций.

Интерфейсы используемые коллекциями:

IEnumerable<T> и **IEnumerable** – предоставляют возможность перечислить её элементы.

ICollection, **IList**, **IDictionary** – предоставляют возможность определения размера коллекции, доступа к элементу по индексу, поиска элемента и модификации коллекции.

IEqualityComparer<T> и **IEqualityComparer** – интерфейсы предоставляют возможность сравнения объектов и проверки объектов на равенство.

Возможности предоставляемые коллекциями:

- Встроенные функции.
- Сортировки.
- Индексирования.
- Автоматическое управление памятью (расширение).
- Синхронизация при доступе к элементам.

Коллекции-списки

Класс **List<T>** из пространства имён **System.Collections.Generic** – это основной класс для представления обычных списков.

Упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза.

```
List<T> listOfTypes = new List<T>() { T, T, T }; // где T – любой тип.
```

Класс **ArrayList** является необобщенной коллекцией, потому что в этой коллекции хранятся элементы разного типа.

Могут быть сложности с обработкой элементов коллекции.

```
ArrayList arrNumbers = new ArrayList();
arrNumbers.Add(1);
arrNumbers.Add(“Hello!");
```

Коллекции-списки

Класс **LinkedList<T>** служит для представления двусвязного списка.

Такой список позволяет осуществлять вставку и удаление элемента без сдвига остальных элементов.

Здесь ссылки в каждом узле указывают на предыдущий и на последующий узел в списке. По двусвязному списку можно передвигаться в любом направлении - как к началу, так и к концу.

```
   LinkedList <T> listOfTypes = new LinkedList <T>() { T, T, T
}; // где T – любой тип.
```

Коллекции-словари

Класс **Dictionary<TKey,TValue>** представляет собой классический словарь с возможностью указать тип для ключа и тип для значения.

Словарь хранит объекты, которые представляют пару ключ-значение.

Каждый такой объект является объектом структуры **KeyValuePair<TKey, TValue>.** Благодаря свойствам **Key** и **Value**, которые есть у данной структуры, мы можем получить ключ и значение элемента в словаре.

Ключ должен иметь уникальное значение.

```
Dictionary dictionary = new Dictionary<string, int>();
dictionary.Add("Cheese", 50);
dictionary.Add("Vine", 40);
```

Коллекции-словари

В **C# 5.0** мы могли инициализировать словари следующим образом:

```
    Dictionary<string, string> countries = new Dictionary<string,
string>
    {
        {"Франция", "Париж"},
        {"Германия", "Берлин"},
        {"Великобритания", "Лондон"}
    };

    foreach(var pair in countries)
        Console.WriteLine("{0} - {1}", pair.Key, pair.Value);
```

Начиная с **C# 6.0** доступен также еще один способ инициализации:

```
    Dictionary<string, string> countries = new Dictionary<string,
string>
    {
        ["Франция"]= "Париж",
        ["Германия"]= "Берлин",
        ["Великобритания"]= "Лондон"
    };
```

Стэк

Класс **Stack<T>** представляет коллекцию, которая использует алгоритм **LIFO** ("последний вошел – первый вышел"). При такой организации каждый следующий добавленный элемент помещается поверх предыдущего. Извлечение из коллекции происходит в обратном порядке – извлекается тот элемент, который находится выше всех в стеке.

Основные методы, которые позволяют управлять элементами:
Метод **Push** – добавляет элемент в стек на первое место.
Метод **Pop** – извлекает и возвращает первый элемент из стека.
Метод **Peek** – просто возвращает первый элемент из стека без его удаления.

```
Stack goods = new Stack();

goods.Push("Cheese");
goods.Push("Vine");

Console.WriteLine(goods.Pop()); // Vine
Console.WriteLine(goods.Pop()); // Cheese
```

Очередь

Класс **Queue<T>** представляет обычную очередь, работающую по алгоритму **FIFO** ("первый вошел – первый вышел").

Основные методы, которые позволяют управлять элементами:
Метод **Dequeue** – извлекает и возвращает первый элемент очереди.
Метод **Enqueue** – добавляет элемент в конец очереди.
Метод **Peek** – просто возвращает первый элемент из начала очереди без его удаления.

```
Queue myQ = new Queue();
myQ.Enqueue("Hello");
myQ.Enqueue("World");

Console.WriteLine(myQ.Dequeue()); // Hello
Console.WriteLine(myQ.Dequeue()); // World
```

Хэш-таблица

Класс **Hashtable** предназначен для создания коллекции, в которой для хранения ее элементов служит хеш-таблица.

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

При хешировании для определения уникального значения, называемого хеш-кодом, используется содержимое специального ключа.

Полученный 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.

```
Hashtable ht = new Hashtable();

ht.Add("alex85", "12345");
ht.Add("fg230404", "12ldsd");
```

Пользовательские коллекции

Иногда требуется создать собственный тип-коллекцию.

Например, в случае, когда изменение коллекции должно генерировать событие, или в случае, когда необходима дополнительная проверка данных при помещении их в коллекцию.

Для этого используется универсальный класс **Collection<T>.**

```
   public class Collection<T> : IList<T>, ICollection<T>,
IEnumerable<T>,
                        IList, ICollection, IEnumerable
   {
      protected IList<T> Items { get; }
      protected virtual void ClearItems();
      protected virtual void InsertItem(int index, T item);
      protected virtual void RemoveItem(int index);
      protected virtual void SetItem(int index, T item);
   }
```

Задания

Задание 1

Создайте 2 списка. Первый типа **ArrayList** и второй **List<T>** - где «Т» будет любым из базовых типов по вашему выбору.

Добавьте в каждый из списко 2 новых элемента.
Удалите элемент с индексом 3.
Удалите из списка 1 добавленный вами элемент.
Отсорируйте полученный список.

После каждого изменения списка выводите его на экран.

Задание 2

Создайте пользовательский список.

Реализуйте добавление в список нового объекта так, чтобы после добавления список сортировался по одному из свойств вашего класса.

Заполните список начальными данными.
Добавьте в ваш список новый элемент.

После каждого изменения списка выводите его на экран.