# Лекция № 2

IT Education Academy

WWW.ITEA.UA

# C# Advanced

## Урок № 2

## Коллекции

# План урока

- Понятие коллекции
- Применение коллекций
- Коллекция ArrayList, List<T> и Dictionary<TKey, TValue>
- Инициализаторы коллекций
- Интерфейсы, которые поддерживаются различными коллекциями
- Ключевое слово yield
- Циклическая конструкция foreach
- Создание пользовательских коллекций
- Основные типы сложных структур данных: список, очередь, стек, словарь, множество

# Понятие коллекции

В языке C# есть **массивы**, которые хранят в себе наборы однотипных объектов, но работать с ними не всегда удобно.

**Минусы использования массивов**

- Массив хранит фиксированное количество объектов, однако что если мы заранее не знаем, сколько нам потребуется объектов.

**Плюсы использования коллекций**

- Плюс коллекций состоит в том, что некоторые из коллекций реализуют стандартные структуры данных, например, стек, очередь, словарь, которые могут пригодиться для решения различных специальных задач.

# Понятие коллекции

- В C# **коллекция** представляет собой совокупность объектов.
- **Коллекция** – это класс, предназначенный для группировки связанных объектов, управления ими и обработки их в циклах.
- **Коллекции** являются важным инструментом программиста, но решение о их применении не всегда оказывается очевидным.


**коллекция алкоголя**

# Применение коллекций

**Коллекции стоит применять, если**:

- Отдельные элементы используются для одинаковых целей и одинаково важны.
- На момент компиляции число элементов неизвестно или не зафиксировано.
- Необходима поддержка операции перебора всех элементов.
- Необходима поддержка упорядочивания элементов.
- Необходимо использовать элементы из библиотеки, от которой потребитель ожидает наличия типа коллекции.

# Коллекция ArrayList

- **ArrayList** представляет коллекцию с динамическим увеличением размера до нужного значения. объектов. И если надо сохранить вместе разнотипные объекты - строки, числа и т.д., то данная коллекция подходит для этого подходит.
- Коллекция **ArrayList** – использует boxing/unboxing, по этому не рекомендуется ее использовать в больших коллекциях.
- При добавлении элементов в коллекцию **ArrayList** ее емкость автоматически увеличивается нужным образом за счет перераспределения внутреннего массива.

```
static void Main()
{
   ArrayList arrayList = new ArrayList();
   arrayList.Add(10);
   arrayList.Add("string");
}
```

# Коллекция List<T>

- Коллекция **List<T>** представляет простейший список однотипных объектов.
- Для работы с коллекциями **Generic** должно быть подключено пространство имен System.Collections.Generic.
- Коллекция **List<T>** предоставляет доступ к элементам по индексу.

```
static void Main()
{
    List<int> list = new List<int>();
    list.Add(10);     // добавление элемента
    list.Add(20);
}
```

# Коллекция Dictionary<TKey, TValue>

- **Dictionary<TKey, TValue>** - словарь который хранит объекты, которые представляют пару ключ - значение.

- В метод Add передаются два параметра: ключ и значение.

```
static void Main()
{
    Dictionary<int, string> dictionary = new Dictionary<int, string>();
    dictionary.Add(0, "Zero"); // добавление элемента
    dictionary.Add(1, "One");
    dictionary.Add(2, "Two");
    dictionary.Add(3, "Three");
}
```

# Инициализаторы коллекций

- В C# имеется специальное средство, называемое **инициализатором коллекции** и упрощающее инициализацию некоторых коллекций.
- Вместо явного вызова метод Add() , при создании коллекции можно указать список инициализаторов. После этого компилятор организует автоматические вызовы метода Add() , используя значения из этого списка.

```
static void Main()
{
   List<int> list = new List<int>();
   list.Add(10);
   list.Add(20);
}
```

или

```
List<int> list = new List<int>() { 10,20 };
```

# Интерфейсы, которые поддерживаются различными коллекциями

- Основой для создания всех коллекций является реализация интерфейсов **IEnumerator** и **IEnumerable**

- Интерфейс **IEnumerator** представляет перечислитесь, с помощью которого становится возможен последовательный перебор коллекции, например, в цикле foreach.

- Интерфейс **IEnumerable** через свой метод GetEnumerator предоставляет перечислитель всем классам, реализующим данный интерфейс. Поэтому интерфейс IEnumerable (IEnumerable<T>) является базовым для всех коллекций.

# Интерфейсы, которые поддерживаются различными коллекциями

## Интерфейс IEnumerable

```
public interface IEnumerable
{
    IEnumerator GetEnumerator();
}
```

Интерфейс **IEnumerable** имеет метод **IEnumerator GetEnumerator()** – который возвращает перечислитель, который можно использовать для навигации по коллекции.

# Интерфейсы, которые поддерживаются различными коллекциями

**Интерфейс IEnumerator -** определяет функционал для перебора внутренних объектов в контейнере

```
public interface IEnumerator
{
   bool MoveNext(); // перемещение на одну позицию вперед в контейнере элементов
   object Current { get; }  // текущий элемент в контейнере
   void Reset(); // перемещение в начало контейнера
}
```

**Свойства интерфейса IEnumerator:**

- `object Current { get; }` – возвращает текущий элемент коллекции.

**Методы интерфейса IEnumerator:**

- `bool MoveNext()` – перемещает перечислитель на следующий элемент коллекции.
- `void Reset()` –возвращает перечислитель на начало коллекции.

# Циклическая конструкция foreach

- Цикл **foreach** предназначен для перебора элементов в контейнерах, в том числе в массивах. Формальное объявление цикла foreach.
- Циклическая конструкция foreach позволяет выполнять навигацию по коллекции, используя реализации интерфейсов Ienumerable и IEnumerator.

```
foreach (var item in collection)
{

}
```

Переменная итерации

- Локальная переменная с неявным типом имеет строгую типизацию, как если бы тип был задан явно, только тип определяет компилятор.

# Ключевое слово yield

- Блок, в котором содержится ключевое слово **yield**, расценивается компилятором, как блок итератора.

- Ключевое слово **return** используется для предоставления значения объекту перечислителя.

- Ключевое слово **break** используется для обозначения конца итерации.

```
public IEnumerator GetEnumerator()
{
   yield return "Hello world!";
}
```

```
public IEnumerator GetEnumerator()
{
   yield break;
}
```

# Очередь Queue

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

**Основные методы класса Queue<T> :**

- Для помещения элементов в коллекцию предназначен метод **Enqueue**().
- При извлечении элементов при помощи метода **Dequeue**() происходит фактическое удаление из коллекции.
- Получить значение элемента без удаления его из коллекции можно при помощи метода **Peek**().

# Стек Stack

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 класса Stack<T> :**

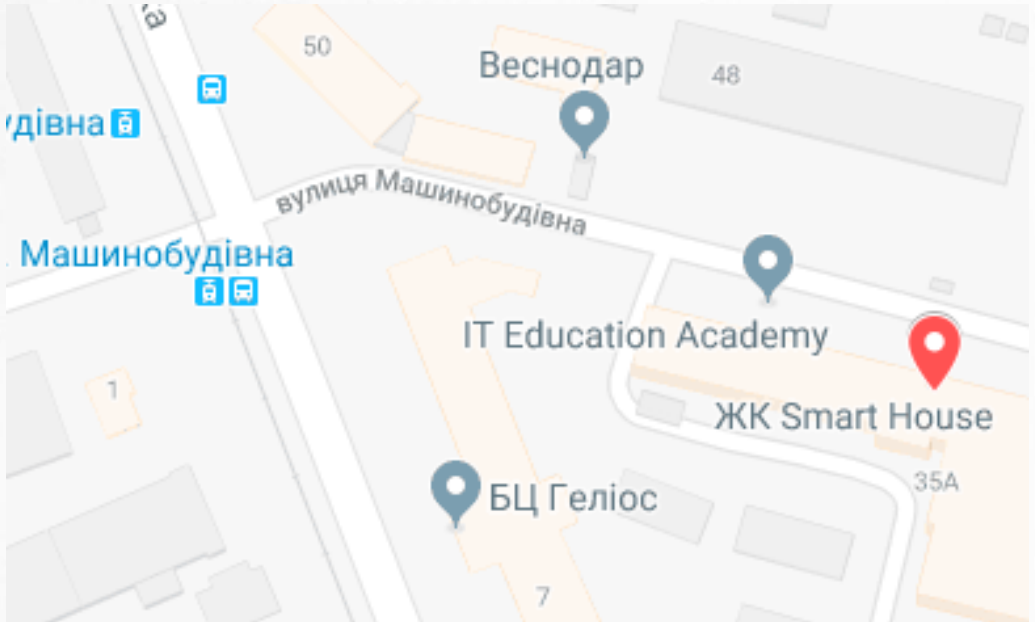
- Для помещения элементов в коллекцию предназначен метод Push().
- При извлечении элементов при помощи метода Pop() происходит фактическое удаление из коллекции.
- Получить значение элемента без удаления его из коллекции можно при помощи метода Peek().

# Хэш таблица Hashtable

Класс **Hashtable** — это структура данных, представляющая собой специальным образом организованный набор элементов хранимых данных. Все данные хранятся в виде пар хеш-значения. Данная структура похожа на словарь, но имеет особенности такие как применение хеш-функции для увеличения скорости поиска.

- Доступ к элементам можно осуществлять по ключам.
- Хранимая информация требует уникальности хэш-кодов, что означает невозможность хранения одинаковых значений.
- Не рекомендуется к использованию, если размер коллекции будет менее 10 элементов.


Презентация окончена.
Спасибо за внимание!





I T E A

# КОНТАКТНЫЕ ДАННЫЕ

**ITEA**

ЖК “Smart House”, ул. Машиностроительная, 41 (м.Берестейская)

ЖК «Корона» улица Срибнокильская,1 м. Позняки

+38 (044) 599-01-79
facebook.com/Itea
info@itea.ua
**itea.ua**