# Занятие 8, ч1. коллекции, обощения

Тренер: Алексей Дышлевой

# Основные вопросы

- Колекції. Типи колекцій. ICollection<Type> та ICollection.
- Класи колекцій
- Списки. Словники. Набори. IEnumerable та IEnumerator. Ініціалізатори колекцій.
- Ітератори. Прямі, зворотні й двоспрямовані ітератори.
- Узагальнені типи. Користувацькі узагальнені типи.
- Ефективність та безпека узагальнень
- Узагальнені класи й структури. Узагальнені інтерфейси. Узагальнені методи. Узагальнені делегати.
- Перетворення узагальненого типу. Значення за замовчуванням.
- Наслідування при узагальненні
- Обмеження
- Узагальнені колекції
- Узагальнені системні інтерфейси

# Что такое коллекция

- Коллекцией является тип, агрегирующий объекты. Коллекция ведет себя как контейнер для набора объектов. Можно создавать экземпляр класса коллекции и добавлять объекты в коллекцию. Можно получить доступ к этим элементам с помощью методов, предоставляемых классом коллекцией.
- Коллекции похожи на массивы. НО!
- При определении массива необходимо указать его размер и тип хранимых в нем данных. Массив типобезопасен, однако имеет ряд ограничений: при определении массива нужно указать количество элементов, которые он может содержать. Часто это значение невозможно определить заранее. При этом, если указать слишком большой размер массива, будет использоваться слишком много памяти, если указать слишком маленький – памяти может не хватить. Массивы работают хорошо, когда точно известно, сколько значений необходимо в нем хранить.

# Коллекции

- При создании экземпляра класса коллекции не нужно указывать размер коллекции.
- При использовании коллекции тип данных для хранения не указывается. Классы коллекций хранят ссылки на другие объекты с помощью типа System.Object. Эта возможность позволяет строить коллекции, хранящие смешанные типы.
- Коллекции не имеют размерностей. Однако, можно имитировать многомерную коллекцию с помощью хранения коллекции в коллекции.

# Интерфейсы классов коллекций

| Интерфейс | Назначение |
|---|---|
| ICollection | Определяет общие характеристики (т.е. размер, перечисление и безопасность к потокам) всех необобщенных типов коллекции |
| ICloneable | Позволяет реализующему объекту возвращать копию самого себя вызывающему коду |
| IDictionary | Позволяет объекту необобщенной коллекции представлять свое содержимое в виде пары «имя/значение» |
| IEnumerable | Возвращает объект, реализующий интерфейс IEnumerator |
| IEnumerator | Позволяет итерацию в стиле foreach по элементам коллекции |
| IList | Обеспечивает поведение добавления, удаления и индексирования элементов в списке объектов |

# Члены ICollection

- CopyTo - этот метод позволяет копировать содержимое коллекции в массив.
- GetEnumerator - этот метод возвращает объект, называемый Enumerator, используемый для перебора элементов коллекции.
- Count - это свойство, которое показывает текущее количество элементов в коллекции

# Пример - ArrayList

- ArrayList list = new ArrayList();
- list.Add(1);
- list.Add(2);
- list.Add(3);
- list.Add(“Some String");
- list.Remove(3); // element
- list.RemoveAt(1); // number of element
- int temp = ((int)list[0]) * 5; //indexer and object casting

# Итерация по коллекции

- Используется оператор foreach
- ArrayList somelist = new ArrayList();
- somelist.Add(25);
- somelist.Add(100);
- somelist.Add(3);
- //...
- foreach (int i in somelist) // i – управляющая переменная
- {
- Console.WriteLine(i);
- }
- Для управляющей переменной важно указать тот же тип, что и тип данных в коллекции; компилятор автоматически генерирует код для приведения к этому типу данных, извлекаемых из коллекции,. Если для управляющей переменной указать неправильный тип, код будет генерировать исключение времени выполнения InvalidCastException

# Основные классы коллекции - ArrayList

- **Класс коллекция ArrayList.** Класс коллекция ArrayList похож на массив. В массив можно добавлять и извлекать элементы с использованием индексирования с нуля. Класс ArrayList динамически увеличивается в размерах по мере добавления значений в коллекцию. Чтобы получить или установить текущий размер коллекции можно использовать свойство Capacity. Класс ArrayList автоматически не сжимается при удалении элементов из коллекции. При удалении значительного числа элементов из коллекции можно использовать метод TrimToSize для уменьшения размера коллекции, или установить свойство Capacity в меньшее значение.

# Основные классы коллекции - ArrayList

- ArrayList intlist = new ArrayList();

- intlist.Add(1);
- intlist.Add(2);
- intlist.Add(3);
- intlist.Add(4);
- intlist.Remove(3);
- intlist.RemoveAt(2);

- int valueFromArrList = (int)intlist[1];

# Основные классы коллекции - Queue

- Класс Queue это FIFO структура данных. Вместо того предоставления методов Add и Remove, класс Queue обеспечивает методы Enqueue и Dequeue. При использовании для объекта метода Enqueue он автоматически добавляется в конец коллекции; при для объекта использовании метода Dequeue, он автоматически удаляется из начала коллекции. Для получения первого элемента очереди, при этом не удаляя его, используется метод Peek. Класс Queue растет автоматически при добавлении объектов в коллекцию. Для восстановления памяти для объекта Queue, за счет уменьшения размера коллекции можно использовать метод TrimToSize.

# Основные классы коллекции - Queue

- Queue somequeue = new Queue();

- somequeue.Enqueue(1);
- somequeue.Enqueue(2);
- somequeue.Enqueue(3);
- somequeue.Enqueue(4);

- int valueFromQueue = (int)somequeue.Dequeue();

# Основные классы коллекции - Stack

- Класс Stack это FILO структура данных. Для добавления и удаления элементов класс Stack предоставляет методы Push и Pop. При использовании метода Push для добавления объекта в коллекцию Stack объект добавляется в начало коллекции, при использовании метода Pop – автоматически извлекается из начала коллекции. Как класс Queue класс Stack предоставляют метод Peek для извлечения элемента из начала коллекции Stack, не удаляя его. Класс Stack растет автоматически, по мере того как объекты добавляются в коллекцию. Если нужно восстановить память для объекта Stack за счет снижения размера коллекции можно использовать метод TrimToSize.

# Основные классы коллекции - Stack

- Stack somestack = new Stack();
- somestack.Push(1);
- somestack.Push(2);
- somestack.Push(3);
- somestack.Push(4);
- int valueFromStack = (int)somestack.Pop();

- int peekValueFromStack = (int)somestack.Peek();

# Задача

- Создать класс Студент с данными о студенте.
- Создать последовательность студентов с помощью классов ArrayList, Queue, Stack.
- Продемонстрировать разницу этих классов в работе сданными.

# Основные классы коллекции - Hashtable

- Класс Hashtable позволяет хранить пары ключ-значение в коллекции быстрого доступа. При добавлении элемента в класс Hashtable с помощью метода Add предоставляются как ключ, так и значение. Ключ коллекции должен быть уникальным, а значение может существовать в двух и более экземплярах. Класс Hashtable хранит объекты, основанные на хэш-значениях ключа.
- Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение играет роль индекса. Из класса Hashtable извлекается значение с помощью индексации и указав ключ значения, которое нужно получить. Класс Hashtable хэширует ключ для определения местоположения требуемого значения. Класс Hashtable значительно быстрее, чем другие коллекции для извлечения элемента из большой коллекции, потому что для нахождения правильного значения просматривает меньшее число элементов. Однако, для небольших коллекций накладные расходы при генерации хэш-кода каждый раз при добавлении значения к коллекции могут фактически снизить производительность приложения. Таким образом для небольших коллекций следует рассмотреть использование другого класса коллекции.
- Класс Hashtable полагается на создания хэш-значений, добавленных к коллекции, по этой причине нужно добавить ключ в коллекцию, где тип ключа реализует метод GetHashCode. Каждый объект включает в себя реализацию по умолчанию метода GetHashCode, унаследованный от класса System.Object. Однако, часто необходимо добавить более сложный алгоритм хэширования для любых разрабатываемых типов. При разработке алгоритма хэширования для использования с классом коллекцией Hashtable, необходимо использовать (case-insensitive) алгоритм без учета регистра.

# Основные классы коллекции - Hashtable

- Hashtable somehashtable = new Hashtable();
- somehashtable.Add(1, “A");
- somehashtable.Add(2, “B");
- somehashtable.Add(3, “C");
- somehashtable.Add(4, “D");

- somehashtable.Remove(3);
- string valueFromHashtable = (string)somehashtable[1];

# Задача

- Использовать класс Hashtable для создания журнала студентов.

# Другие классы коллекции

- LinkedList
- SortedList
- Dictionary
- SortedDictionary
- HashSet
- SortedSet
- BitArray

# Инициализация коллекции

- ArrayList arrlist = new ArrayList();
- arrlist.Add(1);
- arrlist.Add(2);
- ArrayList arrlist1 = new ArrayList() { 1, 2};
- ArrayList arrlist3 = new ArrayList()
- {
- new Person() {Name=“Ivan", Age =33},
- new Person() {Name=“Oksana", Age =18}
- }

# Ограничения коллекций

- Классы коллекции не позволяют указать тип данных, которые может содержать коллекция, что может привести ко многим проблемам. Следующий код будет компилироваться, но при запуске будет генерироваться исключение InvalidCastException.
- ArrayList names = new ArrayList();
- names.Add(«Ivan");
- ...
- int data = (int)names[0];
- Важная проблема – производительность. Встроенные коллекции все содержат элементы типа System.Object. Для ссылки на любой ссылочный тип с небольшими накладными расходами можно использовать тип Object, но если в коллекции хранятся значимые типы, компилятор генерирует код упаковки данных. При извлечении значимого типа из коллекции компилятор генерирует код распаковки данных. Упаковка и распаковка налагать значительные накладные расходы.

# Обобщенный тип

- Обобщенный тип это тип, специфицирующий один или несколько параметров типа. Параметр типа похож на параметр метода за исключением того, что он определяет тип, а не значение. Тип параметра указывается при определении класса с помощью угловых скобок. При создании экземпляра объекта на основе универсального типа необходимо указать тип для замены параметра типа. .NET Framework определяет обобщенные версии некоторых из классов коллекций в пространстве имен System.Collections.Generics, в том числе тип List.
- public class List<T>
- Тип List представляет строго типизированный список объектов, доступ к которым можно получить с использованием индекса. Параметром типа для коллекции names является string, и, следовательно, в этой коллекции можно хранить только значения string. Если попытаться сохранить в ней другой тип, компилятор выдаст ошибку. Кроме того, извлечь данные из коллекции можно без использования приведения типа.
- List<string> names = new List<string>();
- names.Add(“Ivan”);
- names.Add(“Oksana”);
- string name = names[0];
- names.Add(25);

# Компиляция обобщенных типов и безопасность типов

- Замена параметра типа для указанного типа не просто текстовый механизм замены, компилятор при этом выполняет полную семантическую замену
- Для переменных компилятор генерирует различные версии метода Add: следующую версию метода для переменной names.
- public void Add(string item);
- А для переменной listOfLists компилятор генерирует другую версию метода Add.
- public void Add(List<string> item);
- Нельзя вызвать метод Add с другим объектом типа List и передать параметр, имеющий неправильный тип без генерации ошибки компилятора.

# Задача

- Создать список студентов с помощью обобщенного типа

# Словари

- Подобны хэш-таблицам, но используются для обобщенных типов.
- Dictionary<TKey, TValue>

# Задача

- Создать словарь студентов, где ключ – номер зачетной книжки

# Множества

- Коллекция, которая содержит только отличающиеся элементы.
- Используются HashSet<T> и SortedSet<T>

# Задача

- Создать множество в виде списка студентов.
- Продемонстрировать результат повторного добавления студента в множество.

# Пользовательские обобщенные типы

- Для определения пользовательского обобщенного типа нужно добавить один или несколько параметров типа.
- Для определения параметров типа сразу после имени класса добавляются угловые скобки.
- В угловых скобках можно указать имена для каждого из параметров типа, при этом, можно использовать столько параметров типа, сколько необходимо, перечисляя их через запятую.
- Имена, указанные для параметров типа, используются в качестве конкретных типов в классе.

# Пример

```
class UserCollection<TItem>
{
    TItem[] data;
    int index;
    ...
    public void Add(TItem item)
    {
        ...
        data[index] = item;
        ...
    }
}
...
struct User
{
    ...
}
...
UserCollection <User> userList = new UserCollection <User>();
User user = new User(...);
userList.Add(user);
```

# Инициализация значениями по умолчанию

- Если нужно инициализировать члены класса на основе параметра типа значением по умолчанию, не всегда легко определить, что это должно быть за значение по умолчанию. C# предоставляет ключевое слово default.
- Ключевое слово default порождает значимые значение по умолчанию для типа, указанного в качестве параметра типа. Когда компилятор генерирует код для обобщенного типа, он заменяет конструкцию default значением по умолчанию, зависящим от конкретного типа. Значение для ссылочных типов по умолчанию будет null и нуль для числовых значимых типов. Для структуры ключевое слово default будет инициализировать каждый член структуры нулем или null в зависимости от того, являются они значимым типом или ссылкой.

# Пример

```
class UserCollection<TItem>
{
    TItem[] data;
    int index;
    TItem tempData;
    ...
 public UserCollection()
    {
        this.tempData = default(TItem);
        ...
    }
}
```

# Добавление ограничений

- При определении обобщенного типа может потребоваться ограничить типы, которые могут быть использованы в качестве параметров типа, чтобы гарантировать, что они соответствуют определенным критериям или специфическим требованиям типа.
- В C# можно использовать уточнения для ограничения параметров типа, которые можно использовать в обобщенных классах и других типах. Для добавления уточнение после определения класса используется ключевое слово where.

- class UserCollection<TItem> where Titem: Printable
- Ключевое слово where можно добавить для каждого параметра типа, который определяется, и, более того, можно использовать несколько ограничений для каждого типа параметра. Несколько ограничений  определяются, разделяя их уточнения запятой, как класс, реализующий несколько интерфейсов.

# Ограничения

| Ограничение | Описание |
| --- | --- |
| where T: struct | Аргумент типа должен быть типом значения. Могут быть указаны любые типы значения, кроме Nullable. |
| where T : class | Аргумент типа должны быть ссылочным типом; это относится к любому классу, интерфейсу, делегату или типу массив. |
| where T : new() | Аргумент типа должен иметь public конструктор по умолчанию. Когда ограничение new() используется вместе с другими ограничениями, оно должно быть указано последним. |
| where T : <base class name> | Аргумент типа должны быть наследником указанного базового класса. |
| where T : <interface name> | Аргумент типа должны имплементировать указанный интерфейс. Могут быть указаны несколько ограничении интерфейса. Уточняющий интерфейс также может быть универсальным. |
| where T : U | Тип аргумента, который поставляется для T должен вытекать из аргумента, который поставляется для U. |

# Домашнее задание

- Создать коллекцию файлов из выбранной пользователем папки. Сгруппировать файлы по расширениям. Записать результат в отдельный файл. (в отдельном документе объяснить причины выбора конкретного вида коллекции).

- Создать калькулятор для работы с данными разных типов, используя пользовательские обобщенные типы (реализовать основные операции).
- Наложить ограничения по типах