Визуальное программирование

ЛЕКЦИЯ 4

Содержание лекции

01 Обобщения

02 Коллекции

ОБОБЩЕНИЯ

Boxing & unboxing

Упаковка предполагает преобразование объекта значимого типа к типу object. При упаковке общеязыковая среда CLR обертывает значение в объект типа System.Object и сохраняет его в куче.

Распаковка предполагает преобразование объекта типа object к значимому типу.

Упаковка и распаковка ведут к снижению производительности, так как системе надо осуществить необходимые преобразования.

Type safety

Типобезопасность языка программирования означает безопасность/надёжность его системы типов.

Система типов называется безопасной (safe) или надежной (sound), если в программах, прошедших проверку согласования типов, (well-typed programs или well-formed programs) исключена возможность возникновения ошибок согласования типов во время выполнения.

Ошибка согласования типов или ошибка типизации (type error) представляет собой несогласованность типов компонентов выражений в программе, например попытку использовать целое число в роли функции.

Generics

Обобщения (универсальные типы и методы) позволяют разрабатывать классы и методы, которые откладывают спецификацию одного или нескольких типов до тех пор, пока класс или метод не будут объявлены и экземпляры не будут создаваться клиентским кодом.

Пример

```
public class Item<T>
{
    0 references
    public string Name { get; set; }
    0 references
    public T Parameters { get; set; }
}
```

```
var table = new Item<int[]>()
{
    Name = "Table",
    Parameters = new int[] { 30, 65, 70 }
};

var chair = new Item<string>()
{
    Name = "Chair",
    Parameters = "Brown"
};
```

Generics

Достоинства дженериков:

- возможность многократного использования
- типобезопасность
- эффективность

Универсальные типы наиболее часто используются с коллекциями и методами, которые выполняют с ними операции.

Generics

- Использование дженериков позволяет получить максимально широкие возможности многократного использования кода, обеспечения безопасности типов и повышения производительности.
- Чаще всего универсальные шаблоны используются для создания классов коллекций.
- Библиотека классов .NET содержит несколько универсальных классов коллекций в пространстве имен System.Collections.Generic.
- .NETпозволяет создавать универсальные интерфейсы, классы, методы, события и делегаты.
- Универсальные классы можно ограничить, чтобы они разрешали доступ к методам только для определенных типов данных.
- Сведения о типах, используемых в универсальном типе данных, можно получить во время выполнения с помощью рефлексии.

Использование нескольких дженериков

```
public class User<T, K>
{
    1 reference
    public T Id { get; private set; }
    1 reference
    public string Login { get; private set; }
    1 reference
    public K Password { get; set; }

    1 reference
    public User() { }

    1 reference
    public User(T id, string login, K password)...
}
```

```
var user1 = new User<int, string>(1930, "superproger", "qwerty");
var user2 = new User<int[], char>();
```

Ограничения дженериков

В качестве ограничений можно использовать следующие типы:

- классы
- интерфейсы
- class - универсальный параметр должен представлять класс
- struct - универсальный параметр должен представлять структуру
- new() - универсальный параметр должен представлять тип, который имеет общедоступный конструктор без параметров

Пример

```
public class Item
{
    0 references
    public string Name { get; set; }
    0 references
    public int Price { get; set; }
}

0 references
public class Food : Item
{
    0 references
    public int Weight { get; set; }
}
```

```
public class Shop<TItem> where TItem : Item { }

0 references
public class Busket<T> where T : class, new() { }
```

Пример

```
public class Node<T> where T : IComparable<T>
{
    0 references
    public Node<T> Left { get; set; }

    0 references
    public Node<T> Right { get; set; }

    0 references
    public T Data { get; set; }

    0 references
    public int CompareTo(Node<T> other)...
}
```

```
public class Tree<T> where T : IComparable<T>
{
    0 references
    public Node<T> Root { get; set; }

    4 references
    public void Add(T element) { }

    0 references
    public void Remove(T element) { }
}
```

Пример

```
var tree = new Tree<int>(); //Int32
tree.Add(25); //Int32
tree.Add("123"); //string
tree.Add(Convert.ToInt32(new object())); //object
tree.Add((Int16)17); //Int16
```

Наследование дженериков

Существуют следующие варианты наследования:

- создание класса-наследника, который типизирован тем же типом, что и базовый
- создание обычного необобщенного класса-наследника. В этом случае при наследовании у базового класса надо явным образом определить используемый тип
- типизацию производного класса параметром совсем другого типа, отличного от универсального параметра в базовом классе. В этом случае для базового класса также надо указать используемый тип
- использование универсального параметра из базового класса с применением своих параметров

Пример

```
//1st case
0 references
class Furniture<T> : Item<T> { }

//2nd case
0 references
class Furniture : Item<int[]> { }

//3rd case
0 references
class Food<T> : Item<int[]> { }

//4th case
0 references
class Food<T, K> : Item<T> { }
```

Ковариантность и контравариантность

Эти понятия связаны с возможностью использовать в приложении вместо некоторого типа другой тип, который находится ниже или выше в иерархии наследования.

Имеется три возможных варианта поведения:

- Ковариантность: позволяет использовать более конкретный тип, чем заданный изначально
- Контравариантность: позволяет использовать более универсальный тип, чем заданный изначально
- Инвариантность: позволяет использовать только заданный тип

Ковариантность и контрвариантность

```
public interface IItem<in T> { }

0 references
public interface IFood<out T> { }

0 references
public interface IFurniture<T> { }
```

Используется только для интерфейсов и делегатов.

КОЛЛЕКЦИИ

Collections

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.

Collections

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

- необобщенные
- специальные
- с поразрядной организацией
- обобщенные
- параллельные

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.

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.

System.Collections

ICollection	Определяет размер, перечислители и методы синхронизации для всех неуниверсальных коллекций.
IComparer	Предоставляет метод, который сравнивает два объекта.
IDictionary	Представляет небазовую коллекцию пар "ключ-значение".
IDictionaryEnumerator	Перечисляет элементы неуниверсального словаря.
IEnumerable	Предоставляет перечислитель, который поддерживает простой перебор элементов неуниверсальной коллекции.
IEnumerator	Поддерживает простой перебор по неуниверсальной коллекции.
IEqualityComparer	Определяет методы, поддерживающие сравнение объектов на предмет равенства.
IHashCodeProvider	Предоставляет хеш-код объекта, используя пользовательскую хеш-функцию.
IList	Представляет неуниверсальную коллекцию объектов, к каждому из которых можно получить индивидуальный доступ по индексу.
IStructuralComparable	Поддерживает структурное сравнение объектов коллекции.
IStructuralEquatable	Определяет методы, поддерживающие сравнение объектов на предмет структурного равенства.

System.Collections

ArrayList	Реализует интерфейс IList с помощью массива с динамическим изменением размера по требованию.
BitArray	Управляет компактным массивом двоичных значений, представленных логическими значениями, где значение true соответствует включенному биту (1), а значение false соответствует отключенному биту (0).
CaseInsensitiveComparer	Проверяет равенство двух объектов без учета регистра строк.
CaseInsensitiveHashCodeProvider	Предоставляет хэш-код объекта, используя алгоритм хэширования, при котором не учитывается регистр строк.
CollectionBase	Предоставляет базовый класс abstract для строго типизированной коллекции.
Comparer	Проверяет равенство двух объектов с учетом регистра строк.
DictionaryBase	Предоставляет базовый класс abstract для строго типизированной коллекции пар "ключ-значение".
Hashtable	Представляет коллекцию пар «ключ-значение», которые упорядочены по хэш-коду ключа.
Queue	Представляет коллекцию объектов, основанную на принципе «первым поступил — первым обслужен».

System.Collections

ReadOnlyCollectionBase	Предоставляет базовый класс abstract для строго типизированной неуниверсальной коллекции только для чтения.
SortedList	Предоставляет коллекцию пар "ключ-значение", упорядоченных по ключам. Доступ к парам можно получить по ключу и индексу.
Stack	Представляет простую неуниверсальную коллекцию объектов, работающую по принципу «последним поступил — первым обслужен».
StructuralComparisons	Предоставляет объекты для структурного сравнения двух объектов коллекции.
DictionaryEntry	Определяет пару «ключ-значение», которую можно задать или извлечь.

Специальные

Объявляются в пространстве имен 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.

System.Collections.Specialized

INotifyCollectionChanged	Информирует подписчиков о динамических изменениях, например о добавлении или удалении элемента либо об очистке всего списка.
IOrderedDictionary	Представляет индексированную коллекцию пар «ключ-значение».
NotifyCollectionChangedEventHandler	Представляет метод, обрабатывающий событие CollectionChanged.
CollectionsUtil	Создает коллекции, которые не учитывают регистр строк.
HybridDictionary	Реализует интерфейс IDictionary с помощью класса ListDictionary, когда коллекция небольшая, и переключается на класс Hashtable, когда коллекция увеличивается.
ListDictionary	Реализует интерфейс IDictionary с помощью однонаправленного списка. Рекомендуется для коллекций, которые обычно содержат менее 10 элементов.

System.Collections.Specialized

NameObjectCollectionBase	Предоставляет abstract базовый класс для коллекции связанных ключей String и значений Object, доступ к которым можно получить с помощью ключа или индекса.
NameObjectCollectionBase.KeysCollection	Представляет коллекцию ключей String коллекции.
NameValueCollection	Представляет коллекцию связанных ключей String и значений String, доступ к которым можно получить с помощью ключа или индекса.
NotifyCollectionChangedEventArgs	Предоставляет данные для события CollectionChanged.
OrderedDictionary	Представляет коллекцию пар "ключ-значение", доступ к которым можно получить по ключу и индексу.
StringCollection	Представляет коллекцию строк.
StringDictionary	Реализует хэш-таблицу с ключом и значением, строго типизированными как строки, а не объекты.
StringEnumerator	Поддерживает простой перебор коллекции StringCollection.

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.

System.Collections.Generic

CollectionExtensions	Предоставляет методы расширения для универсальных коллекций.
Comparer<T>	Представляет базовый класс для реализаций универсального интерфейса IComparer<T>.
Dictionary<TKey,TValue>.KeyCollection	Представляет коллекцию ключей в Dictionary<TKey,TValue>. Этот класс не наследуется.
Dictionary<TKey,TValue>.ValueCollection	Представляет коллекцию значений в Dictionary<TKey,TValue>. Этот класс не наследуется.
Dictionary<TKey,TValue>	Представляет коллекцию ключей и значений.
EqualityComparer<T>	Представляет базовый класс для реализаций универсального интерфейса IEqualityComparer<T>.
HashSet<T>	Представляет набор значений.
KeyNotFoundException	Исключение, которое выдается, когда ключ, указанный для доступа к элементу в коллекции, не совпадает ни с одним ключом в коллекции.
KeyValuePair	Создает экземпляры структуры KeyValuePair<TKey,TValue>.

System.Collections.Generic

LinkedList<T>	Представляет двунаправленный список.
LinkedListNode<T>	Представляет узел в LinkedList<T>. Этот класс не наследуется.
List<T>	Представляет строго типизированный список объектов, доступных по индексу. Поддерживает методы для поиска по списку, выполнения сортировки и других операций со списками.
PriorityQueue<TElement,TPriority>.UnorderedItemsCollection	Перечисляет содержимое PriorityQueue<TElement,TPriority>объекта без каких-либо гарантий упорядочения.
PriorityQueue<TElement,TPriority>	Представляет коллекцию элементов, имеющих значение и приоритет. При извлечении из очереди удаляется элемент со значением наименьшего приоритета.
Queue<T>	Представляет коллекцию объектов, основанную на принципе «первым поступил — первым обслужен».

System.Collections.Generic

ReferenceEqualityComparer	IEqualityComparer<T>, который использует равенство ссылок (ReferenceEquals(Object, Object)) вместо равенства значений (Equals(Object)) при сравнении двух экземпляров объекта.
SortedDictionary<TKey,TValue>.KeyCollection	Представляет коллекцию ключей в SortedDictionary<TKey,TValue>. Этот класс не наследуется.
SortedDictionary<TKey,TValue>.ValueCollection	Представляет коллекцию значений в SortedDictionary<TKey,TValue>. Этот класс не наследуется.
SortedDictionary<TKey,TValue>	Представляет коллекцию пар "ключ-значение", упорядоченных по ключу.
SortedList<TKey,TValue>	Представляет коллекцию пар "ключ-значение", упорядоченных по ключу на основе реализации IComparer<T>.
SortedSet<T>	Представляет упорядоченную коллекцию объектов.
Stack<T>	Представляет коллекцию переменного размера экземпляров одинакового заданного типа, обслуживаемую по принципу "последним пришел - первым вышел" (LIFO).

Параллельные

Это обобщенные коллекции, определенные в System.Collections.Concurrent.

Поддерживают многопоточный доступ к коллекции.

System.Collections.Concurrent

BlockingCollection<T>	Предоставляет возможности блокировки и ограничения для потокобезопасных коллекций, реализующих IProducerConsumerCollection<T>.
ConcurrentBag<T>	Представляет потокобезопасную неупорядоченную коллекцию объектов.
ConcurrentDictionary<TKey,TValue>	Представляет потокобезопасную коллекцию пар "ключ-значение", доступ к которой могут одновременно получать несколько потоков.
ConcurrentQueue<T>	Предоставляет потокобезопасную коллекцию, обслуживаемую по принципу «первым поступил — первым обслужен» (FIFO).
ConcurrentStack<T>	Предоставляет потокобезопасную коллекцию, обслуживаемую по принципу «последним поступил — первым обслужен» (LIFO).
OrderablePartitioner<TSource>	Представляет определенный способ разделения упорядочиваемого источника данных на несколько разделов.
Partitioner	Предоставляет общие стратегии создания разделов в массивах, списках и перечисляемых коллекциях.
Partitioner<TSource>	Представляет определенный способ разделения источника данных на несколько разделов.

Перечислитель

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>.

IEnumerable

```
namespace System.Collections
{
    //
    // Summary:
    //     Exposes an enumerator, which supports a simple iteration over a non-generic collection.
    public interface IEnumerable
    {
        IEnumerator GetEnumerator();
    }
}
```

IEnumerator

```
namespace System.Collections
{
    public interface IEnumerator
    {
        object Current { get; }

        bool MoveNext();
        void Reset();
    }
}
```

IEnumerator

- MoveNext() перемещает указатель на текущий элемент на следующую позицию в последовательности. Если последовательность еще не закончилась, то возвращает true. Если же последовательность закончилась, то возвращается false.
- Current возвращает объект в последовательности, на который указывает указатель.
- Reset() сбрасывает указатель позиции в начальное положение.

Каким именно образом будет осуществляться перемещение указателя и получение элементов зависит от реализации интерфейса. В различных реализациях логика может быть построена различным образом.

IEnumerable<T>

```
namespace System.Collections.Generic
{
    public interface IEnumerable<out T> : IEnumerable
    {
        //
        // Summary:
        //     Returns an enumerator that iterates through the collection.
        //
        // Returns:
        //     An enumerator that can be used to iterate through the collection.
        IEnumerator<T> GetEnumerator();
    }
}
```

IEnumerator<T>

```
namespace System.Collections.Generic
{
    public interface IEnumerator<out T> : IEnumerator, IDisposable
    {
        //
        // Summary:
        //     Gets the element in the collection at the current position of the enumerator.
        //
        // Returns:
        //     The element in the collection at the current position of the enumerator.
        T Current { get; }
    }
}
```

Итератор

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 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

ВОПРОСЫ ПО ЛЕКЦИИ