Коллекции

Коллекции

- Понятие коллекции
- Обсуждение существующих коллекций.
- Классы коллекций
  - ArrayList,
  - Hashtable,Stack,
  - Queue,
  - SortedList
- Интерфейсы коллекций
  - IList,
  - IEnumerator, IEnumerable,
  - ICollection,
  - IDictionary, IDictionaryEnumerator
  - IComparer,

В C# ***коллекция*** представляет собой совокупность объектов.

Коллекции в C# реализованы с возможностью динамического изменения своих размеров при вставке либо удалении элементов.

Внимание! Коллекции стандартизируют обработку групп объектов т.к. они разработаны на основе набора четко определенных интерфейсов

Типы в .NET Framework для коллекций могут быть разделены на следующие категории:

- **интерфейсы**, которые определяют стандартные протоколы коллекций;
- готовые к использованию **классы коллекций** (списки, словари и т.д.);
- **базовые классы** для написания коллекций, специфичных для приложений.

В среде .NET Framework основные типы коллекций:

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

Необобщенные коллекции

Важно! Необобщенные коллекции могут служить для хранения данных любого типа, причем в одной коллекции допускается наличие разнотипных данных т.к. они оперируют **данными типа object.**

пространство имен System.Collections

ArrayList
Определяет **динамический массив**, т.е. такой массив, который может при необходимости может увеличивать свой размер.

Hashtable
Определяет **хеш-таблицу** для пар «ключ-значение».

SortedList
Определяет **отсортированный список пар** «ключ-значение»

Queue
Определяет список, действующий по принципу «**первым пришел - первым обслужен**»

Stack
Определяет список, действующий по принципу «**первым пришел - последним обслужен**»

Обобщенные коллекции

Важно! **Обобщенные коллекции** обеспечивают **обобщенную** реализацию нескольких стандартных структур данных (списки, стеки, очереди и словари). В обобщенной коллекции могут храниться только такие элементы данных, которые совместимы по типу с данной коллекцией.

пространство имен System.Collections.Generic.

List<T>
Определяет **динамический массив**, т.е. такой массив, который может при необходимости может увеличивать свой размер.

Dictionary<T keys, T values>
Определяет **хеш-таблицу** для пар «ключ-значение».

Queue<T>
Определяет список, действующий по принципу «**первым пришел - первым обслужен**»

Stack<T>
Определяет список, действующий по принципу «**первым пришел - последним обслужен**»

Специальные коллекции

оперируют данными конкретного типа или же делают это каким-то особым образом

пространство имен System.Collections.Specialized

Коллекция с поразрядной организацией

Коллекция с поразрядной организацией - **BitArray** поддерживает операции над отдельными двоичными разрядами.

пространство имен System.Collections

Параллельные коллекции

поддерживают многопоточный доступ к коллекции.

пространство имен System. Collections. Concurrent

В пространстве имен **System.Collections.ObjectModel** находится ряд классов, поддерживающих создание собственных обобщенных коллекций

Интерфейсы необобщенных коллекций

Интерфейсы определяют **функциональные возможности**, которые являются **общими для всех классов необобщенных коллекций**

Интерфейс	Описание
IList	Определяет коллекцию, доступ к которой можно получить с **помощью индексатора**.
ICollection	Определяет элементы, которые должны **иметь все необобщенные коллекции**
IComparer	Определяет метод Compare() для **сравнения объектов**, хранящихся в коллекции
IDictionary	Определяет коллекцию, состоящую из пар «**ключ-значение**»
IEnumerable	Определяет метод **GetEnumerator()**, предоставляющий **перечислитель** для любого класса коллекции

Интерфейс ICollection

служит **основанием**, на котором построены все необобщенные коллекции

- **Count** - содержит количество элементов, хранящихся в коллекции на данный момент;
- **CopyTo()** - копирует содержимое коллекции в массив target, начиная с элемента, указываемого по индексу startIdx.

  Метод обеспечивает переход от коллекции к стандартному массиву.

```
void CopyTo(Array target, int startIdx)
```

Интерфейс IList

int Add (object value)	Добавляет объект **value** в вызывающую коллекцию. Возвращает индекс, по которому этот объект сохраняется
void Clear( )	Удаляет все элементы из вызывающей коллекции
bool Contains (object value)	Возвращает логическое значение true, если вызывающая коллекция содержит объект value, а иначе - false
int IndexOf (object value)	Возвращает индекс объекта value, если этот объект содержится в вызывающей коллекции. Если же объект value не обнаружен, то метод возвращает значение -1
void Insert (int index, object value)	Вставляет в вызывающую коллекцию объект value по индексу index. Элементы, находившиеся до этого по индексу index и дальше, **смещаются вперед**, чтобы освободить место для вставляемого объекта value
void Remove (object value)	Удаляет первое вхождение объекта value в вызывающей коллекции. Элементы, находившиеся до этого за удаленным элементом, **смещаются назад**.
void RemoveAt (int index)	Удаляет из вызывающей коллекции объект, расположенный по указанному индексу index. Элементы, находившиеся до этого за удаленным элементом, **смещаются назад**

Интерфейс IDictionary

определяется поведение, которое позволяет получать значение элемента **коллекции по уникальному ключу**. Ключ представляет собой объект, с помощью которого извлекается значение. коллекции,

Метод	Описание
void Add (object key, object value)	Добавляет в вызывающую коллекцию пару «ключ - значение», определяемую параметрами key и value
void Clear ()	Удаляет все пары «ключ-значение» из вызывающей коллекции
bool Contains (object key)	Возвращает логическое значение true, если вызывающая коллекция содержит объект key в качестве ключа, в противном случае — логическое значение false
iDictionaryEnumerator GetEnumerator()	Возвращает перечислитель для вызывающей коллекции
void Remove (object key)	Удаляет из коллекции элемент, ключ которого равен значению параметра key

Свойство	Описание
bool isFixedSize { get; }	Принимает логическое значение true, если словарь имеет фиксированный размер
bool isReadOnly { get; }	Принимает логическое значение true, если словарь доступен только для чтения
ICollection Keys { get; }	**Получает коллекцию ключей**
ICollection Values { get; }	**Получает коллекцию значений**

В интерфейсе IDictionary определяется следующий индексатор.

```
object this[object key] { get; set; }
```

Этот индексатор служит для получения и установки значения элемента коллекции, а также для добавления в коллекцию нового элемента.

В качестве индекса в данном случае служит ключ элемента, а не собственно индекс.

Интерфейс IEnumerator

Интерфейс **IEnumerator** определяет базовый низкоуровневый протокол, посредством которого производится проход по элементам (перечисление) коллекции в однонаправленной манере.

В интерфейсе IEnumerator определяются **функции перечислителя**.

Для коллекции типа «ключ-значение» **IDictionaryEnumerator**.

```
public interface IEnumerator
{
    bool MoveNext();
    object Current { get; }
    void Reset();
}
```

Метод/Свойство	Назначение
Метод MoveNext ()	переход к следующему элементу коллекции
Метод Reset ()	возобновляет перечисление с самого начала
Свойство Current	содержит текущий элемент

Интерфейс IEnumerable

Интерфейс IEnumerable является необобщенным и используется для поддержки **перечислителей**. **Благодаря реализации интерфейса IEnumerable можно также получать содержимое коллекции в цикле foreach.**

Реализация интерфейса IEnumerable может понадобиться:

- для поддержки оператора foreach;
- для взаимодействия со всем, что ожидает стандартной коллекции;
- для удовлетворения требований более развитого интерфейса коллекции;
- для поддержки инициализаторов коллекций.

Интерфейс IComparer

В интерфейсе IComparer определяется метод Compare () для сравнения двух объектов.

```
int Compare(object x, object y)
```

- возвращает положительное значение, если значение объекта x больше, чем у объекта y;
- возвращает отрицательное - если значение объекта x меньше, чем у объекта y;
- возвращает нулевое - если сравниваемые значения равны.

Важно! **Данный интерфейс можно использовать для указания способа сортировки элементов коллекции.**

Классы необобщенных коллекций

Класс	Описание	Интерфейс
ArrayList	Определяет динамический массив, т.е. такой массив, который может при необходимости увеличивать свой размер	IList, ICollection, IEnumerable, ICloneable
Hashtable	Определяет хеш-таблицу для пар «ключ-значение»	IDictionary, ICollection, IEnumerable, ICloneable
Queue	Определяет очередь, или список, действующий по принципу «первым пришел — первым обслужен»	ICollection, IEnumerable, ICloneable
SortedList	Определяет отсортированный список пар "ключ-значение"	IDictionary, ICollection, IEnumerable, ICloneable
Stack	Определяет стек, или список, действующий по принципу «первым пришел - последним обслужен»	ICollection, IEnumerable, ICloneable

Класс ArrayList

ArrayList – коллекция с **динамическим** увеличением размера до нужного значения. При добавлении элементов в коллекцию ArrayList ее емкость автоматически увеличивается нужным образом за счет перераспределения внутреннего массива.

Коллекция **ArrayList** использует **boxing/unboxing,** поэтому не рекомендуется ее использовать для хранения типов значений.

Метод	Описание
public virtual void AddRange (Icollection c)	Добавляет элементы из коллекции C в конец вызывающей коллекции типа ArrayList
public virtual int BinarySearch (object value)	Выполняет поиск в вызывающей коллекции значения value. Возвращает индекс найденного элемента. Если искомое значение не найдено, возвращает отрицательное значение. Вызывающий список должен быть отсортирован
public virtual void CopyTo (Array array)	Копирует содержимое вызывающей коллекции в массив array, который должен быть одномерным и совместимым по типу с элементами коллекции
public virtual int IndexOf (object value)	Возвращает индекс первого вхождения объекта value в вызывающей коллекции. Если искомый объект не обнаружен, возвращает значение -1
public virtual void InsertRange (int index, ICollection c)	Вставляет элементы коллекции C в вызывающую коллекцию, начиная с элемента, указываемого по индексу index
public virtual int LastIndexOf (object value)	Возвращает индекс последнего вхождения объекта value в вызывающей коллекции. Если искомый объект не обнаружен, метод возвращает значение -1
public static ArrayList Readonly(ArrayList list)	Заключает коллекцию list в оболочку типа ArrayList, доступную только для чтения, и возвращает результат

Метод	Описание
public virtual void Sort ()	Сортирует вызывающую коллекцию по нарастающей
public virtual object[] ToArray()	Возвращает массив, содержащий копии элементов вызывающего объекта
public virtual Array ToArray(Type type)	Возвращает массив, содержащий копии элементов вызывающего объекта. Тип элементов этого массива определяется параметром type
public virtual void TrimToSize()	Устанавливает значение свойства Capacity равным значению свойства Count
public virtual void SetRange (int index, ICollection c)	Заменяет часть вызывающей коллекции, начиная с элемента, указываемого по индексу index, элементами коллекции C
public virtual void RemoveRange (int index, int count)	Удаляет часть вызывающей коллекции, начиная с элемента, указываемого по индексу index, и включая количество элементов, определяемое параметром count
public virtual void Reverse()	Располагает элементы вызывающей коллекции в обратном порядке

Пример работы с ArrayList простые типы

```
  ArrayList al = new ArrayList();
  Console.WriteLine("Исходное количество элементов: "+al.Count);
// Добавить элементы в коллекцию.
  al.Add('C');   al.Add('A');  al.Add('E');   al.Add('B');
  al.Add('D');   al.Add('F');
// Отобразить содержимое коллекции, используя индексирование.
  for (int i = 0; i < al.Count; i++)     Console.Write(al[i] + " ");
   al.Remove('F');  // удаление элемента
   al.RemoveAt(1);  // удаление по индексу
// Добавить количество элементов, достаточное для
// принудительного расширения коллекции.
   for (int i = 0; i < 20; i++)    al.Add((char)('a' + i));
   Console.WriteLine("Кол-во после добавления: " +  al.Count);
// Изменить содержимое коллекции, используя индексирование.
   al[0] = 'X';  al[1] = 'Y';   al[2] = 'Z';
// Копирование в обычный массив.
   char[] mas = new char[al.Count];    al.CopyTo(mas);
// Проверка наличия элемента "X" в коллекции
   if (al.Contains('X'))
       Console.WriteLine("Элемент \"X\" имеется в коллекции!");
   Console.WriteLine("Индекс элемента  \"Z\"" + al.IndexOf('Z'));
// Очистка коллекции
   al.Clear();
```

```
class Car
{
  public int Num { get; set; }
  public string Name { get; set; }
  public override string ToString()
  {
      return String.Format("{0}.{1}",Num, Name);
  }
}
```

Пример работы с ArrayList (сложные типы)

```
   ArrayList al = new ArrayList();
// Добавить элементы в коллекцию.
   al.Add(new Car { Num = 1, Name = "Ford"});
   al.Add(new Car { Num = 2, Name = "BMW" });
   al.Add(new Car { Num = 3, Name = "Opel" });
   al.Add(new Car { Num = 4, Name = "ВАЗ" });
// Удалить элементы из коллекции
   al.Remove(al[3]);  // удаление элемента (ВАЗ)
   al.RemoveAt(1);    // удаление по индексу (BMW)
// Отобразить содержимое коллекции, используя цикл foreach.
   foreach (Car c in al)    Console.WriteLine(c + " ");
// Добавление в коллекцию набора объектов
    ArrayList al2 = new ArrayList();
    al2.Add(new Car { Num = 5, Name = "Audi" });
    al2.Add(new Car { Num = 6, Name = "Fiat" });
    al.AddRange(al2);
// Изменить содержимое коллекции, используя индексирование.
    al[0] = new Car() { Num=1, Name="New Ford "};
// Наличие элемента в коллекции
Car findCar = new Car { Num = 5, Name = "Audi" };
Console.Write("Имеется ли в коллекции - {0}?\n", findCar);
Console.WriteLine(al.Contains(findCar)); // true/false ?
```

Интерфейс IEnumerable

```
ArrayList al = new ArrayList();
Random ran = new Random();

for (int i = 0; i < 10; i++)
        al.Add(ran.Next(1, 20));

// Используем перечислитель
IEnumerator etr = al.GetEnumerator();
while (etr.MoveNext())
    Console.Write(etr.Current + "\t");

Console.WriteLine("\nПовторный вызов перечислителя: \n");
// Сбросим значение и вновь исользуем перечислитель
// для доступа к коллекции
etr.Reset();
while (etr.MoveNext())
    Console.Write(etr.Current + "\t");
```

Итераторы

Что такое итератор? Синтаксис и примеры использования итераторов.

Для реализации **IEnumerable** потребуется предоставить перечислитель. Это можно сделать одним из следующих путей:

- если класс является оболочкой другой коллекции, вернуть перечислитель внутренней коллекции (**например** `String`);
- через итератор с использованием оператора **yield return**;
- за счет создания экземпляра собственной реализации IEnumerator.

Итератор представляет собой метод, оператор или аксессор, *возвращающий по очереди члены совокупности объектов* от ее начала и до конца.

Для преждевременного прерывания итератора служит оператор **yield**:

yield break;

```
public class MyCollection : IEnumerable
    {
        int[] data = { 1, 2, 3 };
        public IEnumerator GetEnumerator()
        {
            foreach (int i in data)
                yield return i;
        }
    }
```

Пример простого итератора

```
Пример  CarPark
3.Audi
4.Opel
5.Fiat
```

```
class Car
{  public int Num { get; set; }
   public string Name { get; set; }
   public override string ToString()
   {  return String.Format("{0}.{1}", Num, Name); }
}

class CarPark
{
   ArrayList park = new ArrayList();
   public void AddCar(Car newCar)
    {  park.Add(newCar);  }
    // создание итератора
   public IEnumerable NumColl(int NumSt, int NumEnd)
    {
      for (int i = 0; i < park.Count; i++)
        {   if (NumSt <= ((Car)park[i]).Num
               && NumEnd >= ((Car)park[i]).Num)
               yield return park[i];
        }
}}
```

Пример простого итератора

```
CarPark parkCars = new CarPark();
parkCars.AddCar(new Car { Num = 1, Name = "BMV"});
parkCars.AddCar(new Car { Num = 2, Name = "Ford" });
parkCars.AddCar(new Car { Num = 3, Name = "Audi" });
parkCars.AddCar(new Car { Num = 4, Name = "Opel" });
parkCars.AddCar(new Car { Num = 5, Name = "Fiat" });
parkCars.AddCar(new Car { Num = 6, Name = "Volvo" });
 // вызов итератора, возвращающего номера машин от 3 до 5
foreach (Car car in parkCars.NumColl(3,5))
  {
   Console.WriteLine(car);
  }
```

Пример работы IComparable

```
public class Student : IComparable
{ private string fio;
  private double ochenka;
  public Student()
   {  fio = "NoName";   ochenka = 0;       }
  public void print()
   { Console.WriteLine("FIO = {0}  средняя = {1}", fio, ochenka); }
  // Реализация метода CompareTo
  public int CompareTo(Object St)
   {   Student x = (Student)St;
       if (String.Compare(x.fio, this.fio) < 0)
         {   return 1;     }
       else if (String.Compare(x.fio, this.fio) == 0)
         {  if (x.ochenka < this.ochenka)
               return 1;
            else if (x.ochenka == this.ochenka)
                    return 0;
                 else
                    return -1;
         }   else     return -1;
   }
}
```

Класс Hashtable

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

ВАЖНО!!! не поддерживают сортировку.

Метод	Описание
public virtual bool ContainsKey (object key)	Возвращает логическое значение true, если в вызывающей коллекции типа Hashtable содержится ключ key, а иначе — логическое значение false
public virtual bool ContainsValue (object value)	Возвращает логическое значение true, если в вызывающей коллекции типа Hashtable содержится значение value, а иначе — логическое значение false
public virtual IDictionaryEnumerator GetEnumerator()	Возвращает для вызывающей коллекции типа Hashtable перечислитель типа IDictionaryEnumerator

Пример работы с Hashtable

```
// Создать хеш-таблицу.
 Hashtable ht = new Hashtable ();

// Добавить элементы в таблицу.
   ht.Add("здание", "жилое помещение");
   ht.Add("автомашина", "транспортное средство");
   ht.Add("книга", "набор печатных слов");
   ht.Add("яблоко", "съедобный плод");

// Добавить элементы с помощью индексатора,
   ht["трактор"] = "сельскохозяйственная машина";

// Получить коллекцию ключей.
   ICollection c = ht.Keys;

// Использовать ключи для получения значений,
   foreach(string str in c)
      Console.WriteLine(str + ": " + ht[str]);
```

```
здание: жилое помещение
книга: набор печатных слов
трактор: сельскохозяйственная машина
автомашина: транспортное средство
яблоко: съедобный плод
```

Класс SortedList

предназначен для создания коллекции, в которой пары «ключ-значение» хранятся в порядке, отсортированном по значению ключей

Метод	Описание
public virtual bool ContainsKey (object key)	Возвращает логическое значение true, если в вызывающей коллекции типа SortedList содержится ключ key, а иначе — логическое значение false
public virtual bool ContainsValue(object value)	Возвращает логическое значение true, если в вызывающей коллекции типа SortedList содержится значение value, а иначе — логическое значение false
public virtual object GetBylndex(int index)	Возвращает значение, указываемое по индексу index
public virtual object GetKey(int index)	Возвращает значение ключа, указываемое по индексу index
public virtual IList GetKeyList()	Возвращает коллекцию типа SortedList с ключами, хранящимися в вызывающей коллекции типа SortedList
public virtual IList GetValueList()	Возвращает коллекцию типа SortedList со значениями, хранящимися в вызывающей коллекции типа SortedList
public virtual int IndexOfKey(object key)	Возвращает индекс ключа key. Если искомый ключ не обнаружен, возвращается значение -1
public virtual int IndexOfValue(object value)	Возвращает индекс первого вхождения значения value в вызывающей коллекции. Если искомое значение не обнаружено, возвращается значение -1

Пример работы с SortedList

```
Содержимое списка по индексатору.
автомашина: транспортное средство
здание: жилое помещение
книга: набор печатных слов
трактор: сельскохозяйственная машина
яблоко: съедобный плод
```

```
// Создать отсортированный список.
   SortedList sList = new SortedList();
// Добавить элементы в список.
   sList.Add("здание", "жилое помещение");
   sList.Add("автомашина", "транспортное средство");
   sList.Add("книга", "набор печатных слов");
   sList.Add("яблоко", "съедобный плод");
// Добавить элементы с помощью индексатора,
   sList["трактор"] = "сельскохозяйственная машина";

// Получить коллекцию ключей.
   ICollection c = sList.Keys;
// Использовать ключи для получения значений.
   Console.WriteLine("Содержимое списка по индексатору.");
   foreach (string str in c)
       Console.WriteLine(str + ": " + sList[str]);
// Отобразить список, используя целочисленные индексы.
   Console.WriteLine("Содержимое списка по индексам.");
   for (int i = 0; i < sList.Count; i++)
       Console.WriteLine(sList.GetByIndex(i));
```

Очередь (queue) - это коллекция, в которой элементы обрабатываются по схеме «первый вошел, первый вышел» (FIFO). Элемент, вставленный в очередь первым, первым же и читается.

Очередь реализуется классом *Queue* из *System.Collections*

Count	возвращает количество элементов в очереди.
Enqueue()	добавляет элемент в конец очереди.
Dequeue()	читает и удаляет элемент из головы очереди. Если на момент вызова метода Dequeue() элементов в очереди больше нет, генерируется исключение InvalidOperationException.
Peek()	читает элемент из головы очереди, но не удаляет его.
TrimExcess()	изменяет емкость очереди. Метод Dequeue() удаляет элемент из очереди, но не изменяет ее емкости. TrimExcess() позволяет избавиться от пустых элементов в начале очереди.

Пример работы с Queue

```
// Создание коллекции - типа "Очередь"
  Queue q = new Queue();
  foreach (int i in q)        Console.Write(i + " ");
  InsertToEnd(q, 22);  InsertToEnd(q, 65);    InsertToEnd(q, 91);
  GetFirst(q);    GetFirst(q);    GetFirst(q);
    try
    {    GetFirst(q);  }
    catch (InvalidOperationException)
    {   Console.WriteLine("Очередь пуста."); }

static void InsertToEnd(Queue q, int a)
{   q.Enqueue(a) ; // метод для добавления эл-та
    Console.WriteLine("Поместить в очередь: Добавлен(" + a + ")");
    Console.Write("Содержимое очереди: ");
    foreach(int i in q)    Console.Write(i + " ");
}
static void GetFirst( Queue q)
{   int a = (int) q.Dequeue(); // метод для извлечения эл-та
    Console.WriteLine (a);
    Console.Write("Содержимое очереди: ");
    foreach(int i in q)  Console.Write(i + " ") ;
}
```

Стек (stack) - это контейнер, работающий по принципу «последний вошел, первый вышел» (LIFO). Элемент, вставленный в очередь первым, читается последним.

Стек реализуется классом *Stack* из *System.Collections*

Count	возвращает количество элементов в стеке.
Push()	добавляет элемент в вершину стека.
Pop()	удаляет и возвращает элемент из вершины стека. Если стек пуст, генерируется исключение типа InvalidOperationException.
Peek()	возвращает элемент из вершины стека, не удаляя его при этом.
Contains()	проверяет наличие элемента в стеке и возвращает true в случае нахождения его там.

Пример работы с Stack

```
  // Создание коллекции - типа “Стек"
  Stack st = new Stack ();
  foreach(int i in st)   Console.Write(i + " ");
  ShowPush(st, 22);   ShowPush(st, 65) ;   ShowPush(st, 91);
  ShowPop(st);   ShowPop(st);   ShowPop(st);
  try
  {  ShowPop(st); }
  catch (InvalidOperationException)
  {  Console.WriteLine("Стек пуст.");   }

static void ShowPop(Stack st)
{   int a = (int) st.Pop(); // метод для извлечения эл-та
     Console.WriteLine(a) ;
     Console.Write("Содержимое стека: ");
     foreach(int i in st)    Console.Write(i + " ");
}
static void ShowPush(Stack st, int a)
{    st.Push(a); // метод для добавления эл-та
     Console.WriteLine("Поместить в стек: Push(" + a + ")");
     Console.Write("Содержимое стека: ");
     foreach (int i in st)    Console.Write(i + " ");
}
```

Обобщенные коллекции

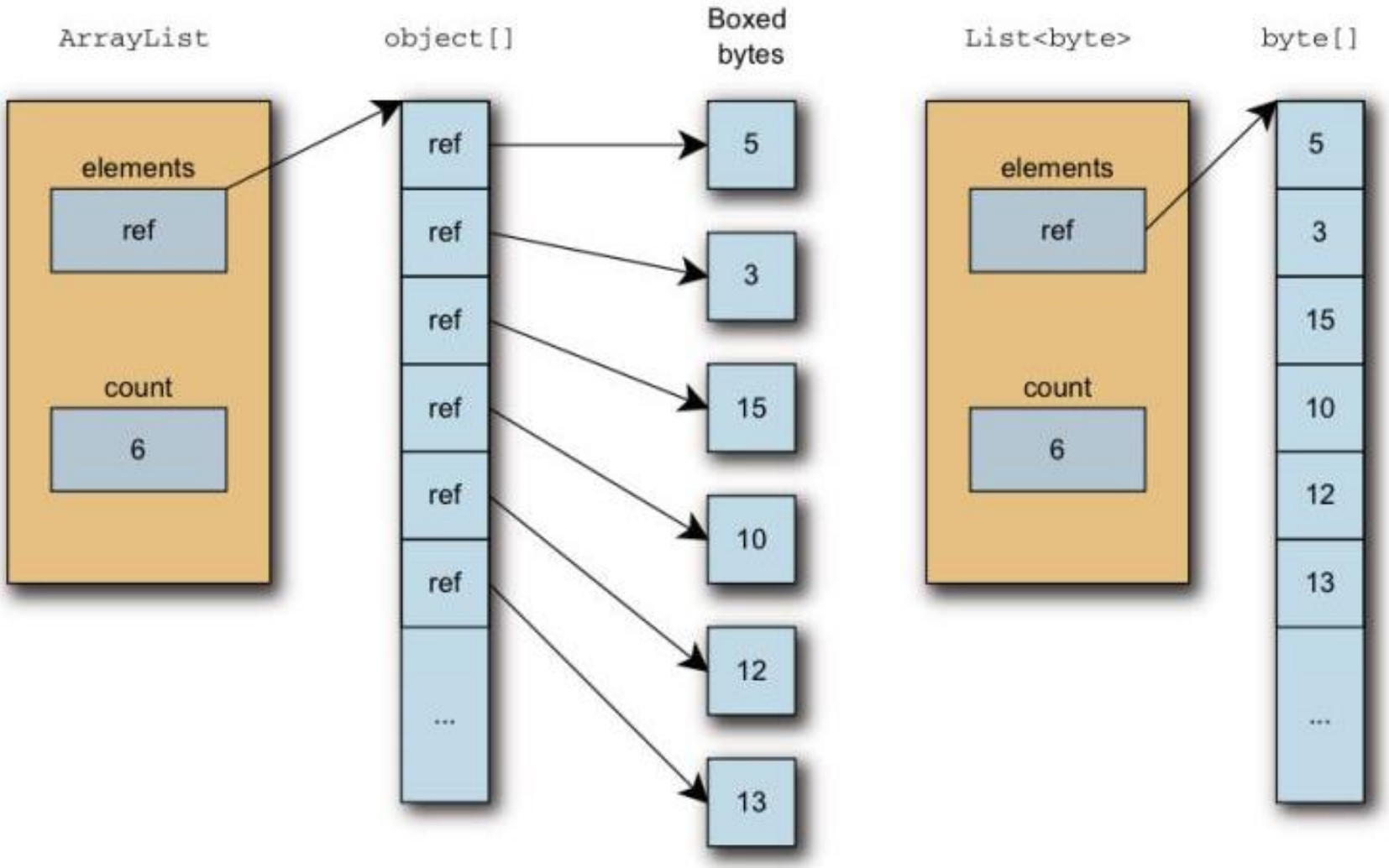
Интерфейсы

Generic интерфейсы	Необобщенные интерфейсы
ILits<T>	ILits
IDictionary<T>	IDictionary
ICollection<T>	ICollection
IEnumerator<T>	IEnumerator
IComparer<T>	IComparer
IComparable<T>	IComparable

Классы коллекций

Generic коллекции	Необобщенные коллекции
Collection<T>	CollectionBase
List<T>	ArrayList
Dictionary<TKey, TValue>	Hashtable
SortedList<TKey, TValue>	SortedList
Stack<T>	Stack
Queue<T>	Queue
LinkedList<T>	Нет

Сравнение List и ArrayList для значимых типов



```
public class List<T>:IList<T>,ICollection<T>,IEnumerable<T>,
                                    IList, ICollection,IEnumerable
    {
        public  List();
        public   void  Add(T item );
        public   Int32 BinarySearch(T item);
        public   void Clear( );
        public   Boolean  Contains(T item);
        public   Int32  IndexOf(T item);
        public   Boolean  Remove (T item);
        public   void  Sort( );
        public   void  Sort(IComparer<T>  compa rer) ;
        public   void  Sort(Comparison<T> comparison);
        public   T[] ToArray( );
        public   Int32  Count  {  get ; }
        public   T this[Int32 index] { get; set;}
    }
```

```
class Car
{
  public string Type {get; set;}    // тип
  public string Name {get; set;}    // название
  public double Speed {get; set;}   // скорость

  public override string ToString()
  {
  return String.Format("{1} {0}, скорость: {2}", Type, Name, Speed);
  }
}
```

```
Console.WriteLine("Обобщенные коллекции!");

// создание обобщенной коллекции List
List<int> listInt=new List<int>();
Random rnd = new Random();
for (int i = 0; i < 10; i++)
    listInt.Add(rnd.Next(100));

Console.WriteLine("Int collection: ");
//при обращении к элементам коллекции возвращается резульат тип int
foreach (int i in listInt)
    Console.Write("{0} ", i);
Console.WriteLine();

List<Car> listCars=
       new List<Car>() {new Car {Name="BMW", Type="X6", Speed=250}};
listCars.Add(new Car{Name="Ford", Type="Focus", Speed=195});

Console.WriteLine("Car collection: ");
//при обращении к элементам коллекции возвращается результат тип Car
foreach (Car i in listCars)
    Console.Write("{0}\n", i);
Console.WriteLine();
```

```
//Словарь для хранения пар:  название группы - количество студентов
Dictionary<string, int> gr = new Dictionary<string, int>();
//добавление значений в список
gr["GR1"] = 12;    gr.Add("GR2", 10);   gr.Add("GR3", 10);
gr.Add("GR4", 6);
//изменение значения
gr["GR1"] = 14;
//вывод всех элементов словаря
foreach (KeyValuePair<string, int> p in gr)
Console.WriteLine("{0} {1}", p.Key, p.Value);
gr.Remove("GR4"); //удаление по значению ключа
//попытка добавления существующего ключа
try
{   gr.Add("GR1", 15); }   catch (ArgumentException e)
                            {Console.WriteLine(e.Message);  }
//попытка обращения к несуществующему ключу
try
{   Console.WriteLine(gr["GR5"]);   }
catch (KeyNotFoundException e)
{ Console.WriteLine(e.Message); }
//проверка существования ключа и получение зачнения
int val;
if (gr.TryGetValue("GR5", out val)) Console.WriteLine(val);
else Console.WriteLine("Key not found");11
```