

## Лабораторная работа № 12 .

### Параметризованные коллекции. Обработка исключительных ситуаций.

**Цель.** Получить практические навыки работы со стандартными коллекциями пространства имен `System.Collection.Generic` и с нестандартными исключениями, создаваемыми программистом.

#### 1. Теоретические сведения

##### 1.1. Параметризованные коллекции.

Во вторую версию библиотеки .NET добавлены параметризованные коллекции для представления основных структур данных, применяющихся при создании программ, — стека, очереди, списка, словаря и т. д. Эти коллекции, расположенные в пространстве имен **System.Collections.Generic**, дублируют аналогичные коллекции пространства имен **System.Collections**.

##### Параметризованные коллекции библиотеки .NET

| Класс-прототип                            | Обычный класс           |
|---|-------------------------|
| <code>Comparer &lt;T&gt;</code>           | <code>Comparer</code>   |
| <code>Dictionary &lt;K,T&gt;</code>       | <code>HashTable</code>  |
| <code>LinkedList &lt;T&gt;</code>         | -                       |
| <code>List &lt;T&gt;</code>               | <code>ArrayList</code>  |
| <code>Queue&lt;T&gt;</code>               | <code>Queue</code>      |
| <code>SortedDictionary &lt;K,T&gt;</code> | <code>SortedList</code> |
| <code>Stack &lt;T&gt;</code>              | <code>Stack</code>      |

Параметром класса-прототипа является тип данных, с которым он работает (T – это тип, который является параметром коллекции, т. е. вместо него можно подставить любой другой тип данных).

В коллекциях, которые мы рассматривали ранее, хранятся ссылки на объекты типа `object`. Когда значение структурного типа приводится к типу `object` (или к типу того интерфейса, который реализован системой) выполняется операция упаковки – явного преобразования из типа значений в тип ссылок. Эта операция выполняется автоматически и не требует вмешательства программиста. Обратной операцией является распаковка – значение объекта присваивается переменной.

У таких коллекций есть два недостатка:

- в одной и той же коллекции можно хранить элементы любого типа, следовательно, ошибки при помещении в коллекцию невозможно проконтролировать на этапе компиляции, а при извлечении элемента требуется его явное преобразование;
- при хранении в коллекции элементов значимых типов выполняется большой объем действий по упаковке и распаковке элементов, что в значительной степени снижает эффективность работы.

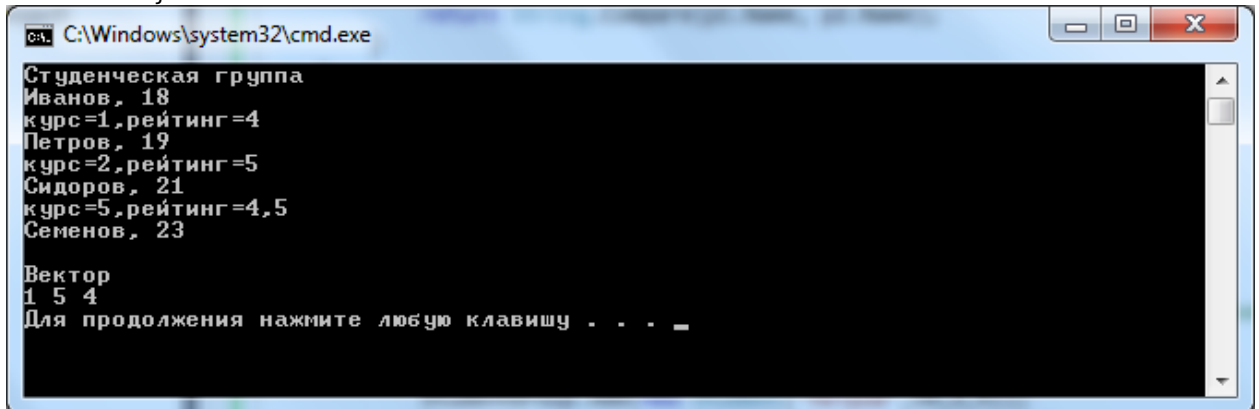
##### Пример 1.

```
class Person : IComparable
{
    . . . .
}
class Student : Person
{
    . . . .
}
class Program
{
    static void Main(string[] args)
    {
        List<Person> StudentGroup = new List<Person>(); //создали коллекцию
        StudentGroup.Add(new Student("Иванов", 18, 1, 4));
        StudentGroup.Add(new Student("Петров", 19, 2, 5));
        StudentGroup.Add(new Student("Сидоров", 21, 5, 4.5));
    }
}
```

```

StudentGroup.Add(new Person("Семенов",23));
Console.WriteLine("Студенческая группа");
foreach (Person x in StudentGroup)
    x.Show();
List<int> Vector = new List<int>();
Vector.Add(1);
Vector.Add(5);
Vector.Add(4);
Console.WriteLine("\nВектор");
foreach (int y in Vector)
    Console.Write(y + " ");
Console.WriteLine();
    }
}

```



Коллекция StudentGroup содержит объекты пользовательских классов иерархии классов Person->Student.

В коллекции, для которой объявлен тип элементов Person, благодаря полиморфизму можно хранить элементы любого производного класса, но не элементы других типов.

Казалось бы, по сравнению с обычными коллекциями это ограничение, а не универсальность, однако на практике коллекции, в которых действительно требуется хранить значения различных, не связанных между собой типов, почти не используются.

Достоинством же такого ограничения является то, что компилятор может выполнить контроль типов во время компиляции, а не выполнения программы, что повышает ее надежность и упрощает поиск ошибок.

Коллекция Vector состоит из целых чисел, причем для работы с ними не требуются ни операции упаковки и распаковки, ни явные преобразования типа при получении элемента из коллекции, как это было в обычных коллекциях.

Классы-прототипы называют также родовыми или шаблонными, поскольку они представляют собой образцы, по которым во время выполнения программы строятся конкретные классы. Использование стандартных параметризованных коллекций для хранения и обработки данных является хорошим стилем программирования, поскольку позволяет сократить сроки разработки программ и повысить их надежность. Рекомендуется тщательно изучить по документации свойства и методы этих классов и выбирать наиболее подходящие в зависимости от решаемой задачи.

#### Параметризованные интерфейсы библиотеки .NET.

| Параметризованный интерфейс | Обычный интерфейс |
|-----------------------------|-------------------|
| ICollection<T>              | ICollection       |
| IComparable<T>              | IComparable       |
| IDictionary<T>              | IDictionary       |
| IEnumerable<T>              | IEnumerable       |
| IEnumerator<T>              | IEnumerator       |
| IList<T>                    | IList             |

## 1.2. Исключения, создаваемые программистом

В C# имеется возможность обрабатывать исключения, создаваемые программистом. Для этого достаточно определить класс как производный от класса Exception. Как правило, определяемые программистом исключения, должны быть производными от класса ApplicationException, "родоначальника" иерархии, зарезервированной для исключений, связанных с прикладными программами. Созданные производные классы не должны ничего реализовывать, поскольку одно лишь их существование в системе типов уже позволит использовать их в качестве исключений.

Классы исключений, создаваемые программистом, будут автоматически иметь свойства и методы, определенные в классе Exception и доступные для них. Конечно, один или несколько элементов в новых классах нужно переопределить.

```
class RangeArrayException : ApplicationException
{
    // Реализуем стандартные конструкторы,
    public RangeArrayException() : base() { }
    public RangeArrayException(string str) : base(str) { }
    // Переопределяем метод ToString() для класса RangeArrayException.
    public override string ToString()
    {
        return Message;
    }
}
class RangeArray
{
    int[] arr; //массив
    int size; //кол-во элементов
    int left_index, right_index; //левая и правая границы массива

    public RangeArray(int left, int right)
    {
        if(right<left) throw new RangeArrayException("Левый индекс больше правого! ");
        arr=new int [right-left+1];
        size=right-left+1;
        left_index=left;
        right_index=right;
    }
    public int Size
    {
        get { return size; }
    }
    //индексатор
    public int this[int index]
    {
        get
        {
            if (index >= left_index && index <= right_index) return arr[index-
left_index];
            else throw new RangeArrayException(" Индекс не попадает в диапазон");
        }
        set
        {
            if (index >= left_index && index <= right_index) arr[index-left_index]
= value;
            else throw new RangeArrayException(" Индекс не попадает в диапазон");
        }
    }
}
class Program
{
    static void Main(string[] args)
```

```

{
    try
    {
        RangeArray a1 = new RangeArray(1, 10);

        Random r = new Random();

        Console.WriteLine("Длина массива a1=" + a1.Size);
        for (int i = 1; i < 10; i++)
        {
            a1[i] = r.Next(0, 100);
            Console.Write(a1[i] + " ");
        }
        Console.WriteLine();

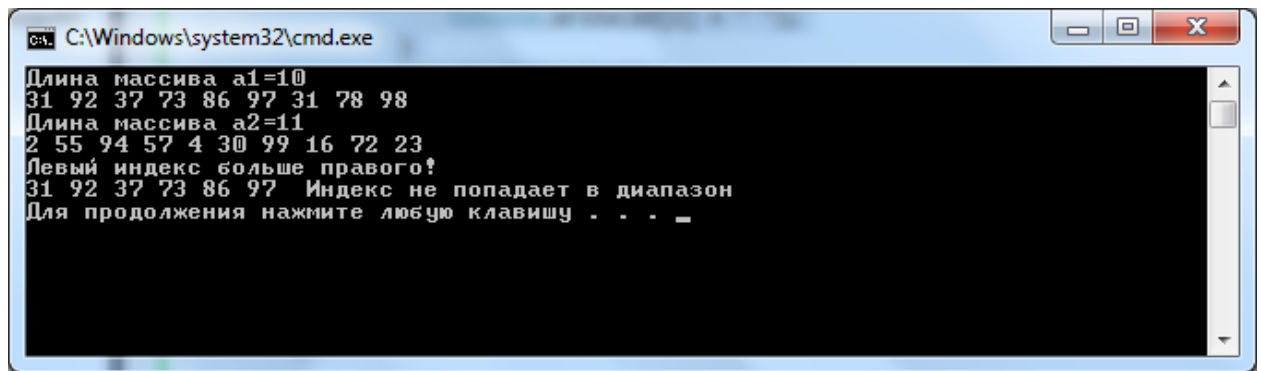
        RangeArray a2 = new RangeArray(-5, 5);
        Console.WriteLine("Длина массива a2=" + a2.Size);
        for (int i = -5; i < 5; i++)
        {
            a2[i] = r.Next(0, 100);
            Console.Write(a2[i] + " ");
        }
        Console.WriteLine();

    }
    catch (RangeArrayException e)
    {
        Console.WriteLine(e);
    }
    try
    { //с ошибкой

        RangeArray a3 = new RangeArray(5, -5);
    }
    catch (RangeArrayException e)
    {
        Console.WriteLine(e);
    }
    try
    { //с ошибкой

        RangeArray a3 = new RangeArray(-5, 5);
        Random r = new Random();
        for (int i = 0; i < 11; i++)
        {
            a3[i] = r.Next(0, 100);
            Console.Write(a3[i] + " ");
        }
        Console.WriteLine();
    }
    catch (RangeArrayException e)
    {
        Console.WriteLine(e);
    }
}
}

```



```
C:\Windows\system32\cmd.exe
Длина массива a1=10
31 92 37 73 86 97 31 78 98
Длина массива a2=11
2 55 94 57 4 30 99 16 72 23
Левый индекс больше правого!
31 92 37 73 86 97 Индекс не попадает в диапазон
Для продолжения нажмите любую клавишу . . .
```

## 2. Постановка задачи

1. Создать иерархию классов (базовый – производный) в соответствии с вариантом (см. лаб. раб. №10).
2. В производном классе определить свойство, которое возвращает ссылку на объект базового класса (это свойство должно возвращать ссылку на объект базового класса, а не ссылку на вызывающий объект производного класса). Например, для иерархии классов Person-Student в классе Student можно определить свойство

```
public Person BasePerson
{
    get
    {
        return new Person(name, age);
    }
}
```
3. Определить класс TestCollections, который содержит поля следующих типов  
Коллекция\_1<TKey>;  
Коллекция\_1<string>;  
Коллекция\_2<TKey, TValue>;  
Коллекция\_2<string, TValue>.  
где тип ключа TKey и тип значения TValue связаны отношением базовый-производный (см. задание 1), Коллекция\_1 и Коллекция\_2 – коллекции из пространства имен System.Collections.Generic.
4. Написать конструктор класса TestCollections, в котором создаются коллекции с заданным числом элементов.
5. Для автоматической генерации элементов коллекций в классе TestCollections надо определить статический метод, который принимает один целочисленный параметр типа int и возвращает ссылку на объект производного типа (Student). Каждый объект (Student) содержит подобъект базового класса (Person). Соответствие между значениями целочисленного параметра метода и подобъектами Person класса Student должно быть взаимно-однозначным.
6. Все четыре коллекции должны содержать одинаковое число элементов. Каждому элементу из коллекции Коллекция\_1<TKey> должен отвечать элемент в коллекции Коллекция\_2<TKey, TValue> с равным значением ключа. Список Коллекция\_1<string> состоит из строк, которые получены в результате вызова метода ToString() для объектов TKey из списка Коллекция\_1<TKey>. Каждому элементу списка Коллекция\_1<string> отвечает элемент в Коллекция\_2 <string, TValue> с равным значением ключа типа string.
7. Для четырех разных элементов – первого, центрального, последнего и элемента, не входящего в коллекцию – надо измерить время поиска элемента в коллекциях Коллекция\_1<TKey> и Коллекция\_1<string> с помощью метода Contains; элемента по ключу в коллекциях Коллекция\_2< TKey, TValue> и Коллекция\_2 <string, TValue > с

помощью метода ContainsKey; значения элемента в коллекции Коллекция\_2< TKey, TValue > с помощью метода ContainsValue.

8. Предусмотреть методы для работы с TestCollections (добавление и удаление элементов).
9. Предусмотреть собственные классы для обработки исключительных ситуаций и выполнить обработку исключений с помощью стандартных исключений, а также с помощью исключений, определенных программистом.

### 3. Варианты

|     | Коллекция_1    | Коллекция_2            |
|-----|----------------|------------------------|
| 1.  | List <T>       | Dictionary <K,T>       |
| 2.  | LinkedList <T> | SortedDictionary <K,T> |
| 3.  | List <T>       | Dictionary <K,T>       |
| 4.  | Queue<T>       | SortedDictionary <K,T> |
| 5.  | Stack <T>      | Dictionary <K,T>       |
| 6.  | List <T>       | SortedDictionary <K,T> |
| 7.  | LinkedList <T> | Dictionary <K,T>       |
| 8.  | List <T>       | SortedDictionary <K,T> |
| 9.  | Queue<T>       | Dictionary <K,T>       |
| 10. | Stack <T>      | SortedDictionary <K,T> |
| 11. | List <T>       | Dictionary <K,T>       |
| 12. | LinkedList <T> | SortedDictionary <K,T> |
| 13. | List <T>       | Dictionary <K,T>       |
| 14. | Queue<T>       | SortedDictionary <K,T> |
| 15. | Stack <T>      | Dictionary <K,T>       |