Теория и практика программирования

Шпилёв Пётр Валерьевич

Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра статистического моделирования

Лекция 6

Санкт-Петербург
2018 г.

e-mail: pitshp@hotmail.com

Упрощенная форма объявления интерфейса

```
interface имя
{
    возвращаемый_тип имя_метода1(список_параметров);
    возвращаемый_тип имя_метода2(список_параметров);
    //...
    возвращаемый_тип имя_методаN(список_параметров);
}
```

Упрощенная форма объявления интерфейса

```
interface имя
{
    возвращаемый_тип имя_метода1(список_параметров);
    возвращаемый_тип имя_метода2(список_параметров);
    //...
    возвращаемый_тип имя_методаN(список_параметров);
}
```

Пример:

```
public interface ISeries
{
    int GetNext(); // возвратить следующее по порядку число
    void Reset(); // перезапустить
    void SetStart(int x); // задать начальное значение
}
```

Упрощенная форма объявления интерфейса

```
interface имя
{
    возвращаемый_тип имя_метода1(список_параметров);
    возвращаемый_тип имя_метода2(список_параметров);
    //...
    возвращаемый_тип имя_методаN(список_параметров);
}
```

Пример:

```
public interface ISeries
{
    int GetNext(); // возвратить следующее по порядку число
    void Reset(); // перезапустить
    void SetStart(int x); // задать начальное значение
}
```

Замечание

Помимо методов, в интерфейсах можно также указывать свойства, индексаторы и события, но не члены данных. В них нельзя также определить конструкторы, деструкторы или операторные методы.

Общая форма реализации интерфейса в классе

```
class имя_класса : имя_интерфейса
{
    // тело класса
}
```

Общая форма реализации интерфейса в классе

```
class имя_класса : имя_интерфейса
{
    // тело класса
}
```

Замечание

В классе можно наследовать базовый класс и в тоже время реализовать один или более интерфейс. В таком случае имя базового класса должно быть указано перед списком интерфейсов, разделяемых запятой.

Общая форма реализации интерфейса в классе

```
class имя_класса : имя_интерфейса
{
    // тело класса
}
```

Упражнение 6.1

Используя интерфейс ISeries, создать класс, генерирующий последовательный ряд чисел, в котором каждое последующее число на два больше предыдущего. Добавить в этот класс метод GetPrevious().

Общая форма реализации интерфейса в классе

```
class имя_класса : имя_интерфейса
{
    // тело класса
}
```

Упражнение 6.1

Используя интерфейс ISeries, создать класс, генерирующий последовательный ряд чисел, в котором каждое последующее число на два больше предыдущего. Добавить в этот класс метод GetPrevious().

Упражнение 6.2

Используя интерфейс ISeries, реализовать класс, генерирующий ряд простых чисел

Общая форма реализации интерфейса в классе

```
class имя_класса : имя_интерфейса
{
    // тело класса
}
```

Упражнение 6.1

Используя интерфейс ISeries, создать класс, генерирующий последовательный ряд чисел, в котором каждое последующее число на два больше предыдущего. Добавить в этот класс метод GetPrevious().

Упражнение 6.2

Используя интерфейс ISeries, реализовать класс, генерирующий ряд простых чисел

Упражнение 6.3

Продемонстрировать интерфейсные ссылки.

Общий вид интерфейсного свойства

```
тип имя_свойства
{
    get; // аксессор без реализации и модификатора доступа
    set; // аксессор без реализации и модификатора доступа
}
```

Общий вид интерфейсного свойства

```
тип имя_свойства
{
    get; // аксессор без реализации и модификатора доступа
    set; // аксессор без реализации и модификатора доступа
}
```

Общая форма объявления интерфейсного индексатора

```
тип_элемента this[int индекс]
{
    get; // аксессор без реализации и модификатора доступа
    set; // аксессор без реализации и модификатора доступа
}
```

Общий вид интерфейсного свойства

```
тип имя_свойства
{
    get; // аксессор без реализации и модификатора доступа
    set; // аксессор без реализации и модификатора доступа
}
```

Общая форма объявления интерфейсного индексатора

```
тип_элемента this[int индекс]
{
    get; // аксессор без реализации и модификатора доступа
    set; // аксессор без реализации и модификатора доступа
}
```

Упражнение 6.4

Добавить в интерфейс ISeries индексатор только для чтения, возвращающий i-й элемент числового ряда.

Пример:

```
// Пример наследования интерфейсов.
using System;
public interface IA
{
    void Meth1();
} //В базовый интерфейс включен метод Meth1().
//а в производный интерфейс добавлен еще один метод — Meth2().
public interface IB : IA
{
    void Meth2();
}
//В этом классе должны быть реализованы методы интерфейсов IA и IB.
class MyClass : IB
{
    public void Meth1()
    {
        Console.WriteLine("Реализовать метод Meth1().");
    }
    public void Meth2()
    {
        Console.WriteLine("Реализовать метод Meth2().");
    }
}
```

Пример: interface IMyIF

```
{
    int MyMeth(int x);
}
class MyClass : IMyIF
{
     int IMyIF.MyMeth(int x) //явная реализация члена интерфейса
    {
         return x / 3;
    }
    public int MyMethI(int x)
    {
         IMyIF a_ob;
        a_ob = this;
        return a_ob.MyMeth(x); // вызов интерфейсного метода IMyIF
    }
}
```

Пример: interface IMyIF

```
{
    int MyMeth(int x);
}
class MyClass : IMyIF
{
     int IMyIF.MyMeth(int x) //явная реализация члена интерфейса
    {
         return x / 3;
    }
    public int MyMethI(int x)
    {
         IMyIF a_ob;
        a_ob = this;
        return a_ob.MyMeth(x); // вызов интерфейсного метода IMyIF
    }
}
```

Упражнение 6.5

Реализовать интерфейс IEven, в котором объявляются два метода. Один будет служить для определения четности числа, второй — для нечетности. Реализовать этим методы в классе MyClass и продемонстрировать их работу. Один из методов нужно реализовать явно.

Общая форма объявления структуры:

```
struct имя : интерфейсы
{
// объявления членов
}
```

Общая форма объявления структуры:

```
struct имя : интерфейсы
{
// объявления членов
}
```

Замечание (1)

Структура, подобна классу, но относится к типу значения, а не к ссылочному типу данных. Структуры не могут наследовать другие структуры и классы или служить в качестве базовых для других структур и классов. (Разумеется, структуры, как и все остальные типы данных в C#, наследуют класс object.)

Общая форма объявления структуры:

```
struct имя : интерфейсы
{
// объявления членов
}
```

Замечание (1)

Структура, подобна классу, но относится к типу значения, а не к ссылочному типу данных. Структуры не могут наследовать другие структуры и классы или служить в качестве базовых для других структур и классов. (Разумеется, структуры, как и все остальные типы данных в C#, наследуют класс object.)

Замечание (2)

Объект структуры может быть создан с помощью оператора new таким же образом,как и объект класса, но в этом нет особой необходимости. Когда этот оператор не используется, объект по-прежнему создается, хотя и не инициализируется.

Общая форма объявления структуры:

```
struct имя : интерфейсы
{
// объявления членов
}
```

Упражнение 6.6

Придумать и реализовать пример обоснованного использования структуры (вместо класса).

Общая форма объявления перечисления

```
enum имя список_перечисления;
```

Пример:

```
using System;
class EnumDemo
{
    enum Apple{Jonathan, GoldenDel, RedDel, Winesap, Cortland, McIntosh};
    static void Main()
    {
        string[] color = {"красный","желтый","красный","красный",
        "красный","красновато-зеленый"};
        Apple i; // объявить переменную перечислимого типа
        // Использовать переменную i для циклического
        // обращения к членам перечисления.
        for(i = Apple.Jonathan; i <= Apple.McIntosh; i++)
        Console.WriteLine(i + " имеет значение " + (int)i);
        Console.WriteLine();
        // Использовать перечисление для индексирования массива.
        for(i = Apple.Jonathan; i <= Apple.McIntosh; i++)
        Console.WriteLine("Цвет сорта " + i + " — " +
        color[(int)i]);
    }
}
```

Общая форма объявления перечисления

```
enum имя список_перечисления;
```

Общая форма объявления перечисления

```
enum имя список_перечисления;
```

Инициализация перечисления

```
enum Apple Jonathan, GoldenDel, RedDel = 10, Winesap, Cortland,
McIntosh ;
```

Общая форма объявления перечисления

```
enum имя список_перечисления;
```

Инициализация перечисления

```
enum Apple Jonathan, GoldenDel, RedDel = 10, Winesap, Cortland,
McIntosh ;
```

Указание базового типа перечисления

```
enum имя : целочисленный_тип список_перечисления ;
```

Общая форма объявления перечисления

enum имя список_перечисления;

Инициализация перечисления

enum Apple Jonathan, GoldenDel, RedDel = 10, Winesap, Cortland, McIntosh ;

Указание базового типа перечисления

enum имя : целочисленный_тип список_перечисления ;

Упражнение 6.7

Сымитировать управление лентой конвейера. Для этой цели можно создать метод Conveyor(), принимающий в качестве параметров следующие команды: "старт","стоп","вперед"и "назад".

Обработка исключений

Обработка исключительных ситуаций в C# организуется с помощью четырех ключевых слов: try, catch, throw и finally.

Обработка исключений

Обработка исключительных ситуаций в C# организуется с помощью четырех ключевых слов: try, catch, throw и finally.

Общая форма определения блоков try/catch

```
try
{
    // Блок кода, проверяемый на наличие ошибок.
}
catch (ExcepType1 exOb)
{
    // Обработчик исключения типа ExcepType1.
}
catch (ExcepType2 exOb)
{
    // Обработчик исключения типа ExcepType2.
}
```

Обработка исключений

Обработка исключительных ситуаций в C# организуется с помощью четырех ключевых слов: try, catch, throw и finally.

Общая форма определения блоков try/catch

```
try
{
    // Блок кода, проверяемый на наличие ошибок.
}
catch (ExcepType1 exOb)
{
    // Обработчик исключения типа ExcepType1.
}
catch (ExcepType2 exOb)
{
    // Обработчик исключения типа ExcepType2.
}
```

Упражнение 6.8

Продемонстрировать обработку исключения типа IndexOutOfRangeException (выход за границы массива)

Обработка исключений

Обработка исключительных ситуаций в C# организуется с помощью четырех ключевых слов: try, catch, throw и finally.

Перехват всех исключений

```
try
{
    // Блок кода, проверяемый на наличие ошибок.
}
catch
{
    // Обработчик всех исключений.
}
```

Упражнение 6.8

Продемонстрировать обработку исключения типа IndexOutOfRangeException (выход за границы массива)

Обработка исключений

Обработка исключительных ситуаций в C# организуется с помощью четырех ключевых слов: try, catch, throw и finally.

Перехват всех исключений

```
try
{
    // Блок кода, проверяемый на наличие ошибок.
}
catch
{
    // Обработчик всех исключений.
}
```

Упражнение 6.9

Использовать вложенный блок try для контроля неучтенных исключений.

Упражнение 6.8

Продемонстрировать обработку исключения типа IndexOutOfRangeException (выход за границы массива)

Общая форма генерирования исключений

```
throw exceptOb;
```

Общая форма генерирования исключений

```
throw exceptOb;
```

Пример:

```
// Сгенерировать исключение вручную.
using System;
class ThrowDemo
{
    static void Main()
    {
        try
        {
            Console.WriteLine("До генерирования исключения.");
            throw new DivideByZeroException();
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Исключение перехвачено.");
        }
        Console.WriteLine("После пары операторов try/catch.");
    }
}
```

Общая форма совместного использования блоков try/ catch и finally

```
try
{
    // Блок кода, предназначенный для обработки ошибок.
}
catch (ExcepType1 exOb) {
    // Обработчик исключения типа ExcepType1.
}
catch (ExcepType2 exOb)
{
    // Обработчик исключения типа ExcepType2.
}
finally
{
    // Код завершения обработки исключений.
}
```

Класс Exception

В классе Exception определяется ряд свойств. К числу самых интересных относятся три свойства, доступные только для чтения: Message, StackTrace и TargetSite:

а) свойство Message содержит символьную строку, описывающую характер ошибки;

б) свойство StackTrace — строку с вызовами стека, приведшими к исключительной ситуации

в) свойство TargetSite получает объект, обозначающий метод, сгенерировавший исключение

Класс Exception

В классе Exception определяется ряд свойств. К числу самых интересных относятся три свойства, доступные только для чтения: Message, StackTrace и TargetSite:

а) свойство Message содержит символьную строку, описывающую характер ошибки;

б) свойство StackTrace — строку с вызовами стека, приведшими к исключительной ситуации

в) свойство TargetSite получает объект, обозначающий метод, сгенерировавший исключение

Класс Exception

В классе Exception определяется ряд свойств. К числу самых интересных относятся три свойства, доступные только для чтения: Message, StackTrace и TargetSite:

а) свойство Message содержит символьную строку, описывающую характер ошибки;

б) свойство StackTrace — строку с вызовами стека, приведшими к исключительной ситуации

в) свойство TargetSite получает объект, обозначающий метод, сгенерировавший исключение

Класс Exception

В классе Exception определяется ряд свойств. К числу самых интересных относятся три свойства, доступные только для чтения: Message, StackTrace и TargetSite:

а) свойство Message содержит символьную строку, описывающую характер ошибки;

б) свойство StackTrace — строку с вызовами стека, приведшими к исключительной ситуации

в) свойство TargetSite получает объект, обозначающий метод, сгенерировавший исключение

Класс Exception

В классе Exception определяется ряд свойств. К числу самых интересных относятся три свойства, доступные только для чтения: Message, StackTrace и TargetSite:

а) свойство Message содержит символьную строку, описывающую характер ошибки;

б) свойство StackTrace — строку с вызовами стека, приведшими к исключительной ситуации

в) свойство TargetSite получает объект, обозначающий метод, сгенерировавший исключение

Упражнение 6.10

Получить в процессе выполнения программы какое-либо исключение и продемонстрировать для него значения вышеперечисленных свойств (например, со помощью метода WriteLine()).

Конструкторы класса Exception

а) public Exception()

б) public Exception(string сообщение)

в) public Exception(string сообщение, Exception внутреннее_исключение)

г) protected Exception(System.Runtime.Serialization.SerializationInfo информация, System.Runtime.Serialization.StreamingContext контекст)

Конструкторы класса Exception

а) public Exception()

б) public Exception(string сообщение)

в) public Exception(string сообщение, Exception внутреннее_исключение)

г) protected Exception(System.Runtime.Serialization.SerializationInfo информация, System.Runtime.Serialization.StreamingContext контекст)

Конструкторы класса Exception

а) public Exception()

б) public Exception(string сообщение)

в) public Exception(string сообщение, Exception внутреннее_исключение)

г) protected Exception(System.Runtime.Serialization.SerializationInfo информация, System.Runtime.Serialization.StreamingContext контекст)

Конструкторы класса Exception

а) public Exception()

б) public Exception(string сообщение)

в) public Exception(string сообщение, Exception внутреннее_исключение)

г) protected Exception(System.Runtime.Serialization.SerializationInfo информация, System.Runtime.Serialization.StreamingContext контекст)

Конструкторы класса Exception

а) public Exception()

б) public Exception(string сообщение)

в) public Exception(string сообщение, Exception внутреннее_исключение)

г) protected Exception(System.Runtime.Serialization.SerializationInfo информация, System.Runtime.Serialization.StreamingContext контекст)

Таблица 1. Наиболее часто используемые исключения, определенные в пространстве имен System.

Исключение	**Значение**
`ArrayTypeMismatchException`	Тип сохраняемого значения несовместим с типом массива
`DivideByZeroException`	Попытка деления на нуль
`IndexOutOfRangeException`	Индекс оказался за границами массива
`InvalidCastException`	Неверно выполнено динамическое приведение типов
`OutOfMemoryException`	Недостаточно свободной памяти для дальнейшего выполнения программы. Это исключение может быть, например, сгенерировано, если для создания объекта с помощью оператора new не хватает памяти
`OverflowException`	Произошло арифметическое переполнение
`NullReferenceException`	Попытка использовать пустую ссылку, т.е. ссылку, которая не указывает ни на один из объектов

Таблица 1. Наиболее часто используемые исключения, определенные в пространстве имен System.

Исключение	**Значение**
`ArrayTypeMismatchException`	Тип сохраняемого значения несовместим с типом массива
`DivideByZeroException`	Попытка деления на нуль
`IndexOutOfRangeException`	Индекс оказался за границами массива
`InvalidCastException`	Неверно выполнено динамическое приведение типов
`OutOfMemoryException`	Недостаточно свободной памяти для дальнейшего выполнения программы. Это исключение может быть, например, сгенерировано, если для создания объекта с помощью оператора new не хватает памяти
`OverflowException`	Произошло арифметическое переполнение
`NullReferenceException`	Попытка использовать пустую ссылку, т.е. ссылку, которая не указывает ни на один из объектов

Упражнение 6.11

Продемонстрировать обработку исключения NullReferenceException.

Пример:

```
class MyException : Exception
{
  /* Реализовать все конструкторы класса Exception. Такие конструкторы
   просто реализуют конструктор базового класса. А поскольку класс
   исключения MyException ничего не добавляет к классу Exception, то
   никаких дополнительных действий не требуется. */
  public MyException() : base() { }
  public MyException(string str) : base(str) { }
  public MyException(string str, Exception inner) : base(str, inner) { }
  protected MyException(System.Runtime.Serialization.SerializationInfo
  si,System.Runtime.Serialization.StreamingContext sc) : base(si, sc) { }
  //Переопределить метод ToString() для класса исключения MyException.
  public override string ToString()
  {
    return Message;
  }
}
```

Пример:

```
class MyException : Exception
{
  /* Реализовать все конструкторы класса Exception. Такие конструкторы
   просто реализуют конструктор базового класса. А поскольку класс
   исключения MyException ничего не добавляет к классу Exception, то
   никаких дополнительных действий не требуется. */
  public MyException() : base() { }
  public MyException(string str) : base(str) { }
  public MyException(string str, Exception inner) : base(str, inner) { }
  protected MyException(System.Runtime.Serialization.SerializationInfo
  si,System.Runtime.Serialization.StreamingContext sc) : base(si, sc) { }
  //Переопределить метод ToString() для класса исключения MyException.
  public override string ToString()
  {
    return Message;
  }
}
```

Упражнение 6.12

Реализовать класс для создания массива, индексируемого в пределах от -10 до 10. Добавить свое исключение для обработки ошибок при обращении к массиву вне допустимого диапазона.

Замечание

Если требуется перехватывать исключения базового и производного классов, то первым по порядку должен следовать оператор catch, перехватывающий исключение производного класса.

Пример:

```
using System;
class ExceptA : Exception //Создать класс исключения.
{
    public ExceptA(string str) : base(str) { }
    public override string ToString(){ return Message; }
}
class OrderMatters
{
    static void Main()
    {
        for(int x = 0; x < 3; x++)
        {
            try
            {
                if(x==0) throw new ExceptA("Перехват ExceptA");
                else throw new Exception();
            }
            catch (ExceptA exc) {Console.WriteLine(exc);}
            catch (Exception exc) {Console.WriteLine(exc);}
        }
    }
}
```

Две общие формы ключевого слова checked

(1) checked (выражение)

(2) checked

```
{
    // проверяемые операторы
}
```

Две общие формы ключевого слова checked

(1) checked (выражение)

(2)
```
checked
{
    // проверяемые операторы
}
```

Две общие формы ключевого слова unchecked

(1) unchecked (выражение)

(2)
```
unchecked
{
    // проверяемые операторы
}
```

Две общие формы ключевого слова checked

(1) checked (выражение)

(2) checked
```
{
    // проверяемые операторы
}
```

Две общие формы ключевого слова unchecked

(1) unchecked (выражение)

(2) unchecked
```
{
    // проверяемые операторы
}
```

Упражнение 6.13

Написать программу в которой демонстрируется применение ключевых слов checked и unchecked