Обзор языка программирования C#

Гутников С.Е.

Цели создания

C++ очень успешен, но C++ сложный язык.

C# стремится предоставить основные возможности C++ в более удобной и простой для использования форме.

Снижение сложности велось по двум основным направлениям.

Во-первых – отказ от "сложных" средств языка C++, таких, как например, управление памятью вручную.

Цели создания

Второе направление снижения сложности – замена "сложных" средств C++ сходными, но более простыми для понимания конструкциями.

Множественное наследование C++ превратилось в простое наследование плюс интерфейсы. Современные версии C# поддерживают подобные на шаблоны обобщенные классы, которые, по мнению разработчиков C#, проще шаблонов C++.

Кроме того, C# предоставляет обширные библиотеки, позволяющие многим программистам сосредоточиться не на написании нового кода, а на соединении уже имеющихся компонентов.

Отличия от C++

1. ***Отличия в распределении объектов***

   В C++ объекты могут располагаться в статической памяти, в динамической памяти (управляемой куче - heap) и в стеке.

   В C# есть две категории типов данных:
   - типы значений (value types) и
   - ссылочные типы (reference types)

   Типы значений в C# похожи на простые типы C++.

Отличия от C++

Ссылочные типы C# похожи на типы C++ доступные через указатели.

В C# объекты располагаются только:

- в динамической памяти (reference types)

  или

- в стеке (value types).

Отличия от C++

2. ***Отличия в организации библиотек***

   В C++ библиотечные функции и классы находятся в пространстве имён `std` и декларированы в соответствующих файлах заголовков (headers) и подключаются директивой препроцессора `#include`.

   На этапе компоновки в исполняемый файл включается объектные модули с реализацией библиотек.

Отличия от C++

В C# все библиотечные классы и интерфейсы доступны программе и помещены в различные пространства имён (`namespace`).

Чтобы можно было использовать сокращённые имена элементов `namespace`, используется директива `using`.

Препроцессор в C# существенно ограничен. Основные отличия от C/C++:

- директиву `#define` нельзя использовать для объявления значений констант и макросов, как это делается в C и C++;
- отсутствует директива `#include`.

Отличия от C++

3. ***Текстовые стандартные файлы***

   Стандартные файлы консоли C++ `cin`, `cout` и `cerr` в языке C# располагаются в `System.Console - In`, `Out` и `Err соответственно`. В отличие от C++ это всегда текстовые файлы.

4. ***Размер символа***

   Тип `char` в C# соответствует «широким» символам в C++ (т.е. типу `wchar_t`).

Отличия от C++

5. ***Константные строки.***

   Строки класса `string` в C# являются константами, изменять их значения нельзя. Всегда, когда нужно изменить значение строки, создаётся новый объект. Следующий пример напечатает `False`.

```
using System;
namespace cstr {
  class Program {
    static void Main(string[] args) {
      string str = "string";
      string strsave = str;
      str += " contains constant values!";
      // str[0] = 'S'; //error
      Console.WriteLine( str == strsave );
}}}
```

Отличия от C++

6. ***Булевские условия в условных операторах и циклах***

   В C++ можно использовать целочисленные выражения в условных операторах и циклах, в таких случаях в C# обязательно используются булевские выражения.

C++	C#

`int i = 5;`
`while(i--)`
`{`
`    // . . .`
`}`
 	
`//////////////`
`if ( i )`
`{`
`    // . . .`
`}`
	`int i = 5;`
	`while(i--!= 0)`
	`{`
	`    // . . .`
	`}`
	
	`//////////////`
	`if ( i != 0 )`
	`{`
	`    // . . .`
	`}`

Отличия от C++

7. ***Сборка мусора и указатели***

В C# отсутствует оператор `delete`, удаление объектов происходит автоматически. Отсутствуют указатели и операции с ними.

8. ***Нулевая ссылка***

В C++ определен макрос `NULL` для обозначения пустых указателей.

В C# для этих целей определен литерал `null`, представляющий пустую ссылку, которая не ссылается ни на один объект.

Отличия от C++

9. **Реализация структур (struct)**

Структуры C++ мало чем отличаются от классов – только типом доступа по умолчанию.

Структуры C# - это типы значений, которые размещаются в стеке и не поддерживают наследования, хотя по синтаксису очень похожи на классы.

Отличия от C++

10. **Общий предок всех объектов**

В C++ нет какого либо общего предка для всех объектов.

В C# класс `Object` пространства имён `System` является исходным базовым классом для всех классов и корнем иерархии типов.

Пример программы

```
using System;
// Подключили пространство имён

namespace prg_01
{
    class Program
    {
        static void Main(string[] args)
        {
            // Элементарный ввод-вывод
            //
            Console.WriteLine("Ваше имя?");
            string name = Console.ReadLine();
            Console.WriteLine(
                "Приветствуем Вас {0}", name);
```

Пример программы

```
// Конвертация из строки в целые
   Console.WriteLine("Введите целое число:");
   string s = Console.ReadLine();
// Здесь может быть исключение если не число:
   int i = Convert.ToInt32(s);

// Конвертация из строки в действительные
   Console.WriteLine(
        "Введите действительное число:");
   s = Console.ReadLine();
// Здесь м.б.исключение при неверном разделителе:
   double d = Convert.ToDouble(s);
   Console.WriteLine(
          "Вы ввели: " + i + "; " + d );
```

Пример программы

```
Console.WriteLine(
        "Введите имя переменной окружения:");

// Читаем посимвольно
name = "";
// читаем очередной символ,
// проверяем на конец файла и на конец строки
while ((i = Console.Read())!= -1 && i != '\n')
{
    // пропускаем символ возврат каретки
    if (i != '\r')
    {
        // накапливаем символы в строке
        name += (char)i; // приведение типа
    }
}
```

Пример программы

```
            // проверяем что ввели не пустую строку
            if (name.Length > 0)
            {
                s = Environment.GetEnvironmentVariable(name);
                // Проверяем что переменная найдена
                if (s != null)
                {
                    // Печать с параметрами
                    Console.WriteLine(
                            "Переменная {0}={1}", name, s);
                }
            }
        }
    }
}
```

Пример программы

Результат работы программы:

```
Ваше имя?
Сергей
Приветствуем Вас Сергей
Введите целое число:
12345
Введите действительное число:
-12345,67
Вы ввели: 12345; -12345,67
Введите имя переменной окружения:
USERNAME
Переменная USERNAME=Sergey Gutnikov
```

Массивы

C# предлагает два типа массивов – прямоугольные и ступенчатые. Эти типы массивов нельзя смешивать между собой в программе.

Рассмотрим работу с массивами на примере.

Пример также демонстрирует объявление обобщённых методов и работу с ними.

Массивы

```
using System;
// Подключили пространство имён

namespace aint
{
    class Program
    {
        // Обобщённый метод – возвращает одномерный
        // массив типа T
        static T[] GetArray<T>(int cx)
        {
            return new T[cx];
        }
```

Массивы

```
// Обобщённый метод - возвращает двумерный
// массив типа T
  static T[][] GetArray<T>(int cy, int cx)
  {
      T[][] r = new T[cy][];

      for (int i = 0; i < cy; i++)
      {
          r[i] = new T[cx];
      }

      return r;

  }
```

Массивы

```
// Обобщённый метод - возвращает трёхмерный
// массив типа T
static T[][][] GetArray<T>(int cz, int cy, int cx)
  {
      T[][][] r = new T[cz][][];
      for (int i = 0; i < cz; i++)
      {
          r[i] = new T[cy][];
          for (int j = 0; j < cy; j++)
          {
              r[i][j] = new T[cx];
          }
      }
      return r;
  }
```

Массивы

```
// Обобщённый метод – возвращает двумерный
// прямоугольный массив типа T
static void GetArray<T>(
                out T[,] r, int cy, int cx)
{
    r = new T[cy, cx];
}

// Обобщённый метод – возвращает трёхмерный
// прямоугольный массив типа T
static void GetArray<T>(
      out T[, ,] r, int cz, int cy, int cx)
{
    r = new T[cz, cy, cx];
}
```

Массивы

```
static void Main()
{
    int[] pix = GetArray<int>(3);
    Console.WriteLine("pix.Length={0}",
                      pix.Length);

    int[][] piyx = GetArray<int>(5, 3);
    Console.WriteLine(
      "piyx.Length={0}, piyx[0].Length={1}",
       piyx.Length, piyx[0].Length );

    int[][][] pizyx = GetArray<int>(7, 5, 3);
```

Массивы

```
            Console.WriteLine(
                "pizyx.Length={0}, pizyx[0].Length={1}, "+
                "pizyx[0][0].Length={2}",
                 pizyx.Length, pizyx[0].Length,
                 pizyx[0][0].Length );
            int[,] aiyx = null;
            GetArray<int>(out aiyx, 5, 3);
            Console.WriteLine("aiyx.Length={0}",
                                aiyx.Length);
            int[, ,] aizyx = null;
            GetArray<int>(out aizyx, 7, 5, 3);
            Console.WriteLine("aizyx.Length={0}",
                                aizyx.Length);
        }
    }
}
```

Массивы

Результат работы программы:

```
pix.Length=3
piyx.Length=5, piyx[0].Length=3
pizyx.Length=7, pizyx[0].Length=5, pizyx[0][0].Length=3
aiyx.Length=15
aizyx.Length=105
```

Классы. Интерфейсы. Свойства

Определения классов в C# похожи на определения классов в C++ и Java.

Поля класса объявляются с типом подобно тому, как это делается со всеми прочими переменными. Ниже перечислены допустимые модификаторы полей:

```
const
public
protected
internal
private
static
readonly
volatile
```

Классы. Интерфейсы. Свойства

Взаимоисключающие модификаторы управляют доступностью поля и к ним относятся `public`, `protected`, `internal` и `private`.

Модификатор `static` управляет тем, является поле членом типа или членом объектов, созданных из этого типа.

Инициализировать поля во время создания объекта можно различными способами. Простейший способ сделать это — прибегнуть к помощи инициализаторов. Они задаются в точке определения поля и могут применяться как для статических полей, так и для полей экземпляра, например:

```
private int x = 789;
```

Классы. Интерфейсы. Свойства

Модификатор `readonly` позволяет определить поле, которое доступно только для чтения. Осуществлять запись в такое поле можно только при создании объекта.

Внутри конструктора можно сколько угодно раз присваивать значения полям `readonly`. Только внутри конструктора поле `readonly` можно передать другой функции как параметр `ref` или `out`.

Отличие между полями `readonly` и `const` в том, что значение переменной `const` известно на этапе компиляции и не может быть изменено даже в конструкторе.

Классы. Интерфейсы. Свойства

Модификатор `volatile` говорит о том, что поле чувствительно ко времени чтения и записи, т.е. поле может быть в любой момент доступно и модифицировано операционной системой или оборудованием, работающим в этой системе, либо, другим потоком управления.

Классы. Интерфейсы. Свойства

Конструкторы именуются по названию определяющего их класса.

В C# два типа конструкторов **статические конструкторы** и **конструкторы экземпляра**. Класс может иметь только один статический конструктор, не имеющий параметров.

Конструкторы экземпляра вызываются при создании экземпляра класса. У класса может быть несколько конструкторов экземпляра, которые могут быть перегружены (т.е. иметь разные типы параметров).

Классы. Интерфейсы. Свойства

Статические методы вызываются на всем классе, а не на его экземплярах. Статические методы имеют доступ только к статическим членам класса и не имеют ссылки `this`.

Методы экземпляра работают с объектами. Для того чтобы вызвать метод экземпляра, необходимо сослаться на экземпляр класса, определяющего этот метод.

Параметры методов передаются по значению. Для передачи по ссылке используются ключевые слова `ref` и `out`.

Классы. Интерфейсы. Свойства

Свойство выглядит и ведет себя аналогично общедоступному полю. Нотация доступа к свойству такая же, как при доступе к общедоступному полю экземпляра.

Однако свойство не имеет никакого ассоциированного с ним места хранения в объекте, как это присуще полям.

Вместо этого свойство представляет сокращенную нотацию для определения средств доступа (`accessors`) для чтения и записи полей.

Классы. Интерфейсы. Свойства

Модификаторы доступа:

`public`

Член полностью видим как извне области определения, так и внутри этой области. Другими словами, доступ к общедоступному члену вообще не ограничен.

`protected`

Член видим только определяющему его классу и любому классу-наследнику данного класса.

Классы. Интерфейсы. Свойства

`internal`

Член видим везде в пределах содержащей его сборки. Сюда входит определяющий его класс и любая область внутри сборки, но вне данного класса.

`protected internal`

Член видим внутри определяющего его класса и везде внутри сборки. Этот модификатор является комбинацией модификаторов `protected` и `internal` с использованием логической операции "ИЛИ".

Классы. Интерфейсы. Свойства

Член также видим любому классу-наследнику определяющего его класса, независимо от того, находится он в той же сборке или нет.

`private`

Член видим только в определяющем классе, без исключений. Это — наиболее строгая форма доступа, и она принята по умолчанию для членов класса.

Классы. Интерфейсы. Свойства

Интерфейсы

В общих чертах синтаксис интерфейса очень похож на синтаксис класса. Однако каждый его член неявно имеет модификатор `public`. Объявление любого члена интерфейса с каким-нибудь явным модификатором приведет к возникновению ошибки времени компиляции. Интерфейсы могут содержать только методы экземпляра; другими словами, включать статические методы в их определения нельзя. Интерфейсы не содержат реализации, т.е. они абстрактны.

Классы. Интерфейсы. Свойства

Наследование

Синтаксис определения класса уже был показан ранее в примерах. Базовый класс указывается после двоеточия, следующего за именем класса. В C# класс имеет только один базовый класс.

При наследовании класса в методе производного класса часто возникает необходимость вызова метода либо доступа к полю, свойству или индексатору базового класса. Для этой цели предусмотрено ключевое слово `base`.

Классы. Интерфейсы. Свойства

Ключевое слово `base` можно применять подобно любой другой переменной экземпляра, но его можно использовать только внутри блока конструктора экземпляра, метода экземпляра или средства доступа к свойству. Применение его в статических методах не допускается.

C# содержит ключевое слово `sealed` для тех случаев, когда нужно сделать так, чтобы клиент не мог наследовать свой класс от конкретного класса.

Классы. Интерфейсы. Свойства

Примененное к целому классу, ключевое слово `sealed` указывает на то, что данный класс является листовым в дереве наследования.

Ключевое слово `abstract` сообщает компилятору, что назначение данного класса - служить базовым, и потому создавать экземпляры этого класса не разрешено.

Классы. Интерфейсы. Свойства

Рассмотрим пример.

```
using System;

namespace geometry
{
    public class Point : Object
    {
      // Поле связанное со свойством X
        private double _x;
```

Классы. Интерфейсы. Свойства

```
// Свойство X:
public double X
{    get {
          return _x * Scale + Shift; }
     set { _x = value; }
}
// Метод: В начало координат
public void SetToOrigin()
{
     X = 0.0;
}
```

Классы. Интерфейсы. Свойства

```
        // Конструктор по умолчанию:
        public Point()
                : this(0.0)
                 // вызов другого
конструктора
        {
        }
        // Конструктор копирования:
        public Point( Point p)
        {
            X = p.X;
        }
```

Классы. Интерфейсы. Свойства

```
// Ещё один конструктор:
public Point(double v)
{
    X = v;
}
// Статические методы: операции
public static Point Add(
                    Point p, Point p1)
{   // p + p1
    return new Point(p.X + p1.X);
}
```

Классы. Интерфейсы. Свойства

```
public static Point Sub(
                    Point p, Point p1)
{    // p - p1
     return new Point(p.X - p1.X);
}
public static Point Mul(
                    Point p, Point p1)
{    // p * p1
     return new Point(p.X * p1.X);
}
```

Классы. Интерфейсы. Свойства

```
public static Point Div(
                    Point p, Point p1)
{   // p / p1
    return new Point( p1.X == 0.0
                ? 1.0 : p.X / p1.X );
}
public static double Distance(
                    Point p, Point p1)
{   // расстояние между точками
    return Math.Abs(p1.X - p.X);
}
```

Классы. Интерфейсы. Свойства

```
// перегрузка операторов:
public static Point operator +(
                    Point p, Point p1)
{
    return Add(p, p1);
}
public static double operator ^(
                    Point p, Point p1)
{
    return Distance(p, p1);
}
```

Классы. Интерфейсы. Свойства

```
public static Point operator -(
                    Point p, Point p1)
{
    return Sub(p, p1);
}
public static Point operator *(
                    Point p, Point p1)
{
    return Mul(p, p1);
}
```

Классы. Интерфейсы. Свойства

```
public static Point operator /(
                    Point p, Point p1)
{
    return Div(p, p1);
}
// Статические свойства:
// автоматически реализуемые
public static double Shift {
                  get; private set; }
public static double Scale {
                  get; private set; }
```

Классы. Интерфейсы. Свойства

```
// Статический конструктор:
static Point()
{
    Shift = 0.0;
    Scale = 1.0;
}
public static void ShiftScale(
    double Shift, double Scale)
{   // Scale > 0
    Point.Shift = Shift;
```

Классы. Интерфейсы. Свойства

```
            Point.Scale = Scale == 0
                    ? 1.0 : Scale;  }
        //--- Демонст. два способа
перегрузки
        //     метода базового класса
        // 1) override - подмена ссылки
        //     в таблице виртуальных методов
        public override string ToString()
        {    // Преобраз.объекта к строке
            return X.ToString();
        }
```

Классы. Интерфейсы. Свойства

```
        // 2) new - скрытие мет.баз.класса
        //   без изменений в
таб.вирт.методов
        public new bool Equals(Object obj)
        {   // Сравнение объектов на
равенство
            Point p = obj as Point;
            // преобразовать obj к Point
            // null если obj не потомок
Point
            if (p == null)
                return base.Equals(p);
```

Классы. Интерфейсы. Свойства

```
            else
                return p.X == X;
        }
    }

    public class Program
    {
        public static int Main(
                                string[] args)
        {
            Point p = new Point(10.0);
```

Классы. Интерфейсы. Свойства

```
        Point _p= new Point {X = 10.0};
        Point p1 = new Point(1.0);
        Point p2 = p * p1;
    Console.WriteLine("p == {0}", p );
    Console.WriteLine("_p == {0}", _p);
    Console.WriteLine("p1 == {0}", p1);
    Console.WriteLine("p2 == {0}", p2);
        Point.ShiftScale(5, 7);
        Console.WriteLine(
      "After coordinate conversion: " +
```

Классы. Интерфейсы. Свойства

```
                "Shift = {0}, Scale = {1}",
                Point.Shift, Point.Scale);
            Console.WriteLine("p = {0}", p);
            Console.WriteLine("_p = {0}", _p);
            Console.WriteLine("p1 = {0}", p1);
            Console.WriteLine("p2 = {0}", p2);
            Console.WriteLine(
                    "p * p1 = {0}", p * p1);
            return 0;
}}}
```

Классы. Интерфейсы. Свойства

Результаты работы:

```
p == 10
p == 10
p1 == 1
p2 == 10
After coordinate conversion: Shift = 5, Scale = 7
p = 75
p = 75
p1 = 12
p2 = 75
p * p1 = 6305
```

Классы. Интерфейсы. Свойства

Рассмотрим ещё один пример.

Создадим массив с возможностью задавать границы изменения индекса и рассмотрим специальное свойство с названием индексатор.

Реализуем две версии :

- на основе абстрактного класса,
- на базе интерфейса

Классы. Интерфейсы. Свойства

```
using System;
using System.IO;
using System.Text;

namespace arridx
{
    public abstract class CArrayExtra
    {
        public int Length {
                        get; protected set; }
```

Классы. Интерфейсы. Свойства

```
        // Размер массива
        public abstract int this[int idx]
            { get; set; } // Индексатор
        public abstract bool IndexInRange(
              int idx);
            // метод проверки индекса
}
public class ArrayXtra : CArrayExtra
{
    private int _min;
```

Классы. Интерфейсы. Свойства

```
private int _max;
private int[] arr;

public ArrayXtra(int nMin, int nMax)
{
    if (nMax <= nMin)
    {
        throw new
            ArgumentException();
    }
```

Классы. Интерфейсы. Свойства

```
        _min = nMin;
        _max = nMax;
        Length = nMax - nMin + 1;
        arr = new int[Length];
    }

    // abstract -> override
    public override bool IndexInRange(
                                int idx)
    {
```

Классы. Интерфейсы. Свойства

```
        return (idx >= _min &&
                idx <= _max);
    }

    public override int this[int idx]
    {
        get
        {
            if (IndexInRange(idx)
                                == false)
```

Классы. Интерфейсы. Свойства

```
            {    // Возбуждаем исключение
            throw new
            IndexOutOfRangeException();
            }
            return arr[idx - _min];
        }
        set
        {
            if (IndexInRange(idx)
                                == false)
            {    // Возбуждаем исключение
```

Классы. Интерфейсы. Свойства

```
                throw new
                IndexOutOfRangeException();
                }
                arr[idx - _min] = value;
}}}
public interface IArrayExtra
{
    int Length { get; }
    bool IndexInRange(int idx);
    int this[int idx] { get; set; }
}
```

Классы. Интерфейсы. Свойства

```
public class ArrayExtra : IArrayExtra
{
    private int _min;
    private int _max;
    private int[] arr;

    public int Length {
                        get; private set; }
    public ArrayExtra(
                        int nMin, int nMax)
    {
```

Классы. Интерфейсы. Свойства

```
        if (nMax <= nMin)
        {
        throw new ArgumentException();
        }
        _min = nMin;
        _max = nMax;
        Length = nMax - nMin + 1;
        arr = new int[Length];
    }
```

Классы. Интерфейсы. Свойства

```
public bool IndexInRange(int idx)
{
    return (idx >= _min &&
            idx <= _max);
}
public int this[int idx]
{
    get
    {
    if (IndexInRange(idx) == false)
```

Классы. Интерфейсы. Свойства

```
            {
            throw new
            IndexOutOfRangeException();
            }
            return arr[idx - _min];
        }
        set
        {
            if (IndexInRange(idx) ==
                                    false)
            {
```

Классы. Интерфейсы. Свойства

```
                throw new
                IndexOutOfRangeException();
                }
                arr[idx - _min] = value;
            }
        }
    }
    public class AExtra :
                    ArrayXtra, IArrayExtra
    {
```

Классы. Интерфейсы. Свойства

```
        public AExtra(int nMin, int nMax)
            : base(nMin, nMax) { }
    }
    class Program
    {
        public static void TestIndex(
                IArrayExtra pa,
                int nMin, int nMax)
        {
            int i;
```

Классы. Интерфейсы. Свойства

```
// test set
for (i = nMin; i <= nMax; i++){
    pa[i] = i;
}
// test get
for (i = nMin; i <= nMax; i++)
{
    Console.Write(
                " {0}", pa[i]);
}
```

Классы. Интерфейсы. Свойства

```
        Console.WriteLine();
    }

    public static void FindBounds(
        IArrayExtra pa, out int nMin,
                        out int nMax )
    // on start -
    // nMin==any correct index in array
    {
        if (pa.Length <= 0)
```

Классы. Интерфейсы. Свойства

```
{
    throw new
        ArgumentException();
}
int n = Int32.MaxValue;
int d = pa.Length - 2;
while (pa.IndexInRange(n)
                    == false)
{
    n -= d;
}
```

Классы. Интерфейсы. Свойства

```
nMin = n;
try
{
    while (true)
    {
        n = pa[nMin - 1];
        nMin--;
    }
}
catch (Exception e)
```

Классы. Интерфейсы. Свойства

```
{
if ((e is IndexOutOfRangeException)
            == false)
// ( e as IndexOutOfRangeException)
//          == null
          {
              throw e;
          }
     }
     nMax = nMin + pa.Length - 1;
}
```

Классы. Интерфейсы. Свойства

```
public static int Main(
                        string[] args)
{
    ArrayExtra a = null;
    try {
        a = new ArrayExtra(-3, 3);
        TestIndex(a, -3, 3);
        int i, j;

        FindBounds(a, out i, out j);
```

Классы. Интерфейсы. Свойства

```
            Console.WriteLine(
            "Array bounds:[{0}, {1}]",
            i, j);

            // exception:
            a = new ArrayExtra(3, -3);
        }

        catch (Exception e) {
```

Классы. Интерфейсы. Свойства

```
                    Console.Error.WriteLine(
                     "Runtime error:\n{0}",
                     e.ToString());
                }
                return 0;
            }
        }
}
```

Классы. Интерфейсы. Свойства

Результаты работы:

```
 -3 -2 -1 0 1 2 3
Array bounds:[-3, 3]
Runtime error:
System.ArgumentException: Value does not fall within
   the expected range.
    at arridx.ArrayExtra..ctor(Int32 nMin, Int32
   nMax)
    at arridx.Program.Main(String[] args)
```