Лекция 32

Наследование

Наследование

Наследование является одним из трех основополагающих принципов объектно-ориентированного программирования, поскольку оно допускает создание иерархических классификаций.

В языке **C# производный** класс представляет собой специализированный вариант **базового** класса. Он наследует все переменные, методы, свойства и индексаторы, определяемые в базовом классе, добавляя к ним свои собственные элементы.

```
class имя_производного_класса : имя_базового_класса {
    // тело класса
}
```

Для любого производного класса можно указать **только один базовый класс**.

Пример 1

```
namespace WindowsFormsApp1 {
  public partial class Form1 : Form {
    public Form1() {
   InitializeComponent();
   }
   private void button1_Click(object sender, EventArgs e) {
      Triangle t1 = new Triangle();
      Triangle t2 = new Triangle();
      t1.Width = 4.0; t1.Height = 4.0;
      t1.Style = "равнобедренный";
      t2.Width = 8.0; t2.Height = 12.0;
      t2.Style = "прямоугольный";
```

Пример 1

```
  richTextBox1.AppendText("Сведения об объекте t1: "
+ "\n");
  richTextBox1.AppendText(t1.ShowStyle()+"\n");
  richTextBox1.AppendText(t1.ShowDim() + "\n");
  richTextBox1.AppendText("Площадь равна " +
t1.Area() + "\n");

  richTextBox1.AppendText("Сведения об объекте t2: "
+ "\n");
  richTextBox1.AppendText(t2.ShowStyle() + "\n");
  richTextBox1.AppendText(t2.ShowDim() + "\n");
  richTextBox1.AppendText("Площадь равна " +
t2.Area() + "\n");
```

Пример 1

```
  Rectangle r1 = new Rectangle();
  r1.Width = 4.0; r1.Height = 6.0;
  bool c = r1.IsSquare();
  if (c == true)
richTextBox1.AppendText("Прямоугольник
равносторонний" + "\n");
  else  richTextBox1.AppendText("Прямоугольник не
равносторонний" + "\n");
   richTextBox1.AppendText("Площадь равна " +
t1.Area() + "\n");
}
private void button2_Click(object sender, EventArgs e){
Close(); }
}
```

Пример 1

```
class TwoDShape {    // Класс для двумерных объектов
   public double Width;          public double Height;
   public string ShowDim() {
        return "Ширина и высота равны " +
   Width.ToString() + " и " + Height.ToString();
   }
}
 class Triangle : TwoDShape {
// Класс Triangle, производный от класса TwoDShape
    public string Style; // тип треугольника
    public double Area() { return Width * Height / 2; }
   // Площадь треугольника
    public string ShowStyle() { return "Треугольник " +
   Style.ToString(); }  // Тип треугольника
}
```

Пример 1

```
 class Rectangle : TwoDShape { // Класс для
// прямоугольников, производный от класса TwoDShape.
// Возвратить логическое значение true,
// если прямоугольник является квадратом
    public bool IsSquare() {
       if (Width == Height) return true;
           return false;
   }
    public double Area() { // Площадь прямоугольника
         return Width * Height;
    }
  }
}
```

Наследование

Если базовый класс содержит закрытые члены, то в производном классе они недоступны.

Для преодоления такого ограничения в **C#** предусмотрены разные способы.

- использование защищенных (**protected**) членов базового класса;
- применение открытых **свойств** для доступа к закрытым данным базового класса.

С помощью свойства можно ввести ограничения на доступ к значению переменной или же сделать ее доступной только для чтения. Так, если сделать свойство открытым, но объявить его базовую переменную закрытой, то этим свойством можно будет воспользоваться в производном классе, но нельзя будет получить непосредственный доступ к его базовой закрытой переменной.

Пример 2

```
class TwoDShape {
    double pri_width;       // закрытая переменная
    double pri_height;      // закрытая переменная
    public double Width {
    // Свойства ширины и высоты двумерного объекта
        get { return pri_width; }
        set { pri_width = value < 0 ? -value : value; }
    }
    public double Height {
        get { return pri_height; }
        set { pri_height = value < 0 ? -value : value; }
    }
    public void ShowDim() {
         Console.WriteLine("Ширина и высота равны " + Width
  + " и " + Height); }
}
```

Пример 2

```
// Класс для треугольников, производный от класса TwoDShape
class Triangle : TwoDShape {
    public string Style; // тип треугольника
    public double Area() {
      // Площадь треугольника
       return Width * Height / 2;
     }
    public void ShowStyle() {
      // Тип треугольника
       Console.WriteLine("Треугольник " + Style); }
}
```

Пример 2

```
class Shapes2 {
    static void Main() {
        Triangle t1 = new Triangle ();
        Triangle t2 = new Triangle ();
        t1.Width = 4.0;            t1.Height = 4.0;
        t1.Style = "равнобедренный";
        t2.Width = 8.0;            t2.Height = 12.0;
        t2.Style = "прямоугольный";
        Console.WriteLine("Сведения об объекте t1: ");
        t1.ShowStyle();            t1.ShowDim();
        Console.WriteLine ("Площадь равна " + t1.Area());
        Console.WriteLine("Сведения об объекте t2: ");
        t2. ShowStyle();           t2.ShowDim();
        Console.WriteLine("Площадь равна " + t2.Area());
    }
}
```

Наследование

Защищенный член создается с помощью модификатора доступа **protected**. Если член класса объявляется как **protected**, он становится закрытым, но за исключением одного случая, когда защищенный член наследуется. В этом случае защищенный член базового класса становится защищенным членом производного класса, а значит, доступным для производного класса. Таким образом, используя модификатор доступа **protected**, можно создать члены класса, являющиеся закрытыми для своего класса, но все же наследуемыми и доступными для производного класса.

Пример 3

```
class B {
    protected int i, j;     // члены, закрытые для класса B,
                            // но доступные для класса D
    public void Set(int a, int b) {
        i = a;              j = b;
    }
    public void Show() { Console.WriteLine (i + " " + j);  }
}
class D : B {
    int k; // закрытый член
    // члены i и j класса B доступны для класса D
    public void Setk() {   k = i * j;   }
    public void Showk() { Console.WriteLine(k);   }
}
```

Пример 3

```
class ProtectedDemo {
    static void Main() {
        D ob = new D();
        ob.Set(2, 3);  // допустимо, т. к.  доступно для класса D
        ob.Show();     // допустимо, т. к. доступно для класса D
        ob.Setk();     // допустимо, т. к. входит в класс D
        ob.Showk();    // допустимо, т. к. входит в класс D
    }
}
```

Модификатор доступа **protected** следует применять в том случае, если требуется создать член класса, доступный для всей иерархии классов, но для остального кода он должен быть закрытым. А для управления доступом к значению члена класса лучше воспользоваться свойством.

Наследование

В иерархии классов допускается, чтобы у базовых и производных классов были свои собственные **конструкторы**. При этом конструктор базового класса конструирует базовую часть объекта, а конструктор производного класса — производную часть этого объекта.

Если в классе не определен конструктор, то он создается автоматически (по умолчанию). Но на практике конструкторы определяются в большинстве классов.

Если конструктор определен только в производном классе, то все происходит очень просто: конструируется объект производного класса, а базовая часть объекта автоматически конструируется его конструктором, используемым по умолчанию.

Пример 4

```
class TwoDShape {     // Класс для двумерных объектов
    double pri_width;
    double pr.i_height;
    public double Width{// Свойства ширины и длины объекта
        get { return pri_width; }
        set { pri_width = value < 0 ? -value : value; }
    }
   public double Height {
        get { return pri_height; }
        set { pri_height = value < 0 ? -value : value; }
   }
   public void ShowDim() { Console.WriteLine("Ширина и
    длина равны " + Width + " и " + Height); }
}
```

Пример 4

```
class Triangle : TwoDShape { // Класс для треугольников,
    // производный от класса TwoDShape
      string Style;
      public Triangle(string s, double w, double h) {
        // Конструктор
            Width = w; // инициализация членов базового класса
            Height = h; // инициализация членов базового класса
            Style = s; // инициализация членов производного класса
      }
      public double Area() { return Width * Height / 2; }
      public void ShowStyle() {
    Console.WriteLine("Треугольник " + Style); }
}
```

Пример 4

```
class Shapes3 {
    static void Main() {
      Triangle t1 = new Triangle("равнобедренный", 4.0, 4.0);
      Triangle t2 = new Triangle("прямоугольный", 8.0, 12.0);
      Console.WriteLine("Сведения об объекте t1: ");
      t1.ShowStyle();                t1.ShowDim();
      Console.WriteLine ("Площадь равна " + t1.Area());
      Console.WriteLine("Сведения об объекте t2: ");
      t2.ShowStyle();                t2.ShowDim();
      Console.WriteLine("Площадь равна " + t2.Area());
    }
}
```

Наследование

Когда конструкторы определяются как в базовом, так и в производном классе, процесс построения объекта несколько усложняется, поскольку должны выполняться конструкторы обоих классов.

В данном случае приходится обращаться к еще одному ключевому слову языка **C#: base**, которое находит двоякое применение: во-первых, для вызова конструктора базового класса; и во-вторых, для доступа к члену базового класса, скрывающегося за членом производного класса.

```
конструктор_производного_класса (список_параметров)
    : base (список_аргументов) {
        // тело конструктора
}
```

где **список_аргументов** обозначает любые аргументы, необходимые конструктору в базовом классе.

Наследование

Когда в производном классе указывается ключевое слово **base**, вызывается конструктор из его непосредственного базового класса. Следовательно, ключевое слово **base** всегда обращается к базовому классу, стоящему в иерархии непосредственно над вызывающим классом. Это справедливо даже для многоуровневой иерархии классов.

Аргументы передаются базовому конструктору в качестве аргументов метода **base**(). Если же ключевое слово отсутствует, то автоматически вызывается конструктор, используемый в базовом классе по умолчанию.

С помощью ключевого слова **base** можно вызвать конструктор любой формы, определяемой в базовом классе, причем выполняться будет лишь тот конструктор, параметры которого соответствуют переданным аргументам.

Пример 5

```
class TwoDShape {
    double pri_width;          double pri_height;
    public TwoDShape() {
   // Конструктор, вызываемый по умолчанию
        Width = Height = 0.0;
    }
    public TwoDShape(double w, double h) {
    // Конструктор с параметрами класса TwoDShape
        Width = w;          Height = h;
    }
    public TwoDShape(double x) {
   // Сконструировать объект равной ширины и высоты
        Width = Height = x;
    }
```

Пример 5

```
  public double Width {
    // Свойства ширины и высоты объекта
        get { return pri_width; }
        set { pri_width = value < 0 ? -value : value; }
  }
  public double Height {
        get { return pri_height; }
        set { pri_height = value < 0 ? -value : value; }
  }
  public void ShowDim() {
        Console.WriteLine("Ширина и высота равны " +
Width + " и " + Height);
  }
}
```

Пример 5

```
class Triangle : TwoDShape { // производный класс
    string Style;
    public Triangle() { // конструктор по умолчанию
        Style = "null";
    }
    public Triangle(string s, double w, double h) : base(w, h) {
        // конструктор, принимающий три аргумента
        Style = s;
    }
    public Triangle(double x) : base(x) {
        Style = "равнобедренный";
    }
    public double Area() { return Width * Height / 2; }
    public void ShowStyle() {
        Console.WriteLine("Треугольник " + Style);
    }
}
```

Пример 5

```
class Shapes5 {
    static void Main() {
        Triangle t1 = new Triangle();
        Triangle t2 = new Triangle("прямоугольный", 8.0, 12.0);
        Triangle t3 = new Triangle(4.0);
        t1 = t2;
        Console.WriteLine("Сведения об объекте t1: ");
        t1.ShowStyle();          t1.ShowDim();
        Console.WriteLine("Площадь равна " + t1.Area());
        Console.WriteLine("Сведения об объекте t2: ");
        t2.ShowStyle();          t2.ShowDim();
        Console.WriteLine("Площадь равна " + t2.Area());
        Console.WriteLine("Сведения об объекте t3: ");
        t3.ShowStyle();          t3.ShowDim();
       Console.WriteLine("Площадь равна " + t3.Area());
    }
}
```

Наследование

В производном классе можно определить **член с таким же именем**, как и у члена его базового класса. В этом случае член базового класса скрывается в производном классе. И хотя формально в **C#** это не считается ошибкой, компилятор все же выдаст сообщение, предупреждающее о том, что имя скрывается. Если член базового класса требуется скрыть намеренно, то перед его именем следует указать ключевое слово **new**, чтобы избежать появления подобного предупреждающего сообщения. Следует, однако, иметь в виду, что это совершенно отдельное применение ключевого слова **new**, не похожее на его применение при создании экземпляра объекта.

Пример 6

```
class A {
    public int i = 0;
}
class B : A {    // создать производный класс
    new int i;   // этот член скрывает член i из класса A
    public B(int b) {
        i = b;    // член i в классе B
    }
    public void Show() {
       Console.WriteLine("Член i в производном классе: " + i);
    }
}
class NameHiding {
    static void Main() {
        B ob = new B (2);
        ob.Show();
    }
}
```

Наследование

Имеется еще одна форма ключевого слова **base**, которая действует подобно ключевому слову **this**, за исключением того, что она всегда ссылается на базовый класс в том производном классе, в котором она используется: **base**.элемент

где **элемент** - может обозначать метод или переменную экземпляра. Эта форма ключевого слова **base** чаще всего применяется в тех случаях, когда в производном классе нужно обратиться к скрытому (с помощью ключевого слова **new**) одноименному члену базового класса.

Пример 7

```
class A {
   public int i = 0;
   public void Show() {    // Метод Show() в классе A
        Console.WriteLine("Член i в базовом классе: " + i);
  }
}
class B : A {    // Создать производный класс
  new int i;   // этот член скрывает член i из класса A
  public B(int a, int b) {
    base.i = a; // здесь обнаруживается скрытый член из класса A
    i = b;      // член i из класса B
  }
```

Пример 7

```
  new public void Show() {
  // Здесь скрывается метод Show() из класса A
     base.Show();      // вызов метода Show() из класса A
     Console.WriteLine("Член i в производном классе: " + i);
    }
}

class UncoverName {
      static void Main() {
            B ob = new B(1, 2);
            ob.Show ();
      }
}
```

Наследование

В **C#** можно также строить **иерархии**, состоящие из любого числа уровней наследования. Так, если имеются три класса, **A**, **B** и **C**, то класс **C** может наследовать от класса **B**, а тот, в свою очередь, от класса **A**. В таком случае каждый производный класс наследует характерные особенности всех своих базовых классов. В частности, класс **C** наследует все члены классов **B** и **A**.

Пример 8

```
class TwoDShape {
    double pri_width;          double pri_height;
    public TwoDShape() {
    // Конструктор, используемый по умолчанию
        Width = Height = 0.0;
    }
    public TwoDShape(double w, double h) {
   // Конструктор для класса TwoDShape
        Width = w;          Height = h;
    }
     public TwoDShape(double x) {
  // Сконструировать объект равной ширины и высоты
        Width = Height = x;
    }
```

Пример 8

```
    public double Width {
   // Свойства ширины и высоты объекта
          get { return pri_width; }
          set { pri_width = value < 0 ? -value : value; }
    }
    public double Height {
        get { return pri_height; }
        set { pri_height = value < 0 ? -value : value; }
    }
    public void ShowDim() {
         Console.WriteLine("Ширина и высота равны " +
  Width + " и " + Height);
    }
}
```

Пример 8

```
class Triangle : TwoDShape {
   string Style; // закрытый член класса
   public Triangle () { Style = "null"; }
   public Triangle(string s, double w, double h) : base(w, h){
        Style = s;
   }
   public Triangle(double x) : base(x) {
        Style = "равнобедренный";
   }
   public double Area() { return Width * Height / 2; }
   public void ShowStyle() {
        Console.WriteLine("Треугольник " + Style);
   }
}
```

Пример 8

```
class ColorTriangle : Triangle {
    // Расширить класс Triangle
        string color;
        public ColorTriangle(string c, string s, double w,
    double h) : base(s, w, h) {
                color = c;
        }
        public void ShowColor() {
        // Показать цвет треугольника
                Console.WriteLine("Цвет " + color);
        }
}
```

Пример 8

```
class Shapes6 {
    static void Main() {
        ColorTriangle t1 = new ColorTriangle("синий",
    "прямоугольный", 8.0, 12.0);
        ColorTriangle t2 = new ColorTriangle("красный",
    "равнобедренный", 2.0, 2.0);
        Console.WriteLine("Сведения об объекте t1: ");
        t1.ShowStyle();          t1.ShowDim();
    t1.ShowColor();
        Console.WriteLine ("Площадь равна " + t1.Area());
        Console.WriteLine("Сведения об объекте t2: ");
        t2.ShowStyle();          t2.ShowDim();
    t2.ShowColor();
        Console.WriteLine("Площадь равна " + t2.Area());
    }
}
```

Наследование

В иерархии классов конструкторы вызываются по порядку выведения классов: от базового к производному. Более того, этот порядок остается неизменным независимо от использования ключевого слова **base**.

Так, если ключевое слово **base** не используется, то выполняется конструктор по умолчанию, т.е. конструктор без параметров.

Наследование

Членом класса может быть другой класс. В этом случае внешний класс так и называется **внешним** (**outer**), а внутренний называют **вложенным** (**nested**).

Достоинством вложенных классов является то, что им доступны все элементы внешнего класса. Метод вложенного класса может обращаться ко всем закрытым переменным внешнего. Кроме того, вложенный класс можно скрыть от всех других классов, определив его с модификатором **private**.

Наконец, открытый вложенный класс доступен во всей области видимости внешнего класса. Если класс **Outer** внешний, а открытый класс **Nested** вложен в него, то ссылаться на **Nested** следует как на **Outer**.**Nested**, причем внешний класс действует (в определенной степени) как пространство имен.

Вложенные классы приблизительно эквивалентны статическим внутренним классам на языке **Java**.

Пример 9

```
public class F {
   public F(int n, int d) {
       this.n = n;        this.d = d;
   }
    // Методы опущены...
   public override string ToString() {
      StringBuilder s = new StringBuilder();
      s.AppendFormat("{0}/{1}", n, d);
      return s.ToString();
   }
   internal class FA {
     public void Draw(F f) {
        Console.WriteLine("Числитель: {0}", f.n);
        Console.WriteLine("Знаменатель: {0}", f.d);
     }
   }
   private int n;
   private int d;
}
```

Пример 9

```
public class Tester {
      static void Main() {
            F f1 = new F(3,4);
            Console.WriteLine("f1: {0}", f1.ToString());
            F.FA  fa = new F.FA();
            fa.Draw(f1);
      }
}
```

C# является строго типизированным языком программирования. Это означает, что переменная ссылки на объект класса одного типа, как правило, не может ссылаться на объект класса другого типа.

Пример 10

```
class X {
    int a;
    public X(int i) { a = i; }
}
class Y {
    int a;
    public Y(int i) { a = i; }
}
class IncompatibleRef {
    static void Main() {
        X x = new X(10);
        X x2;
        Y y = new Y(5) ;
        x2 = x; // верно, т. к. оба объекта относятся к одному и тому же типу
      //  x2 = y; // ошибка, поскольку это разнотипные объекты
    }
}
```

Наследование

Вообще говоря, переменная ссылки на объект может ссылаться только на объект своего типа.

Но из этого принципа строгого соблюдения типов в **C#** имеется одно важное исключение: **переменной ссылки на объект базового класса может быть присвоена ссылка на объект любого производного от него класса**. Такое присваивание считается вполне допустимым, поскольку экземпляр объекта производного типа инкапсулирует экземпляр объекта базового типа. Следовательно, по ссылке на объект базового класса можно обращаться к объекту производного класса.

Пример 11

```
using System;
class X {
    public int a;
    public X(int i) { a = i; }
}
class Y : X {
    public int b;
    public Y(int i, int j) : base(j) { b = i; }
}
class BaseRef {
    static void Main() {
        X x = new X(10);
        X x2;
        Y y = new Y(5, 6);
         x2 = x; // верно, поскольку оба объекта
                 // относятся к одному и тому же типу
```

Пример 11

```
        Console.WriteLine("x2.a: " + x2.a);
         x2 = y; // тоже верно, поскольку класс Y
                 // является производным от класса X
         Console.WriteLine ("x2.a: " + x2.a);
        // ссылкам на объекты класса X
        // известно только о членах класса X
        x2.a = 19; // верно
        // x2.b = 27; // неверно, поскольку член b
                      // отсутствует у класса X
    }
}
```

Наследование

Один из самых важных моментов для присваивания ссылок на объекты производного класса переменным базового класса наступает тогда, когда конструкторы вызываются в иерархии классов.

В классе нередко определяется **конструктор, принимающий объект своего класса в качестве параметра**. Благодаря этому в классе может быть сконструирована копия его объекта. Этой особенностью можно выгодно воспользоваться в классах, производных от такого класса.

Пример 12

```
class TwoDShape {
    double pri_width;
    double pri_height;
    public TwoDShape() { Width = Height = 0.0; }
    public TwoDShape(double w, double h) {
        Width = w;          Height = h;
    }
    public TwoDShape(double x) {
        Width = Height = x;
    }
    public TwoDShape(TwoDShape ob) {
    // Сконструировать копию объекта TwoDShape
        Width = ob.Width;          Height = ob.Height;
    }
```

Пример 12

```
    public double Width {
    // Свойства ширины и высоты объекта
        get { return pri_width; }
        set { pri_width = value < 0 ? -value : value; }
    }
    public double Height {
       get { return pri_height; }
       set { pri_height = value < 0 ? -value : value; }
    }
    public void ShowDim() {
        Console.WriteLine("Ширина и высота равны " +
  Width + " и " + Height);
    }
}
```

Пример 12

```
class Triangle : TwoDShape {
    string Style;
    public Triangle() { Style = "null"; }
    public Triangle(string s, double w, double h) : base(w, h) {
        Style = s;
     }
    public Triangle(double x) : base(x) {
        Style = "равнобедренный";
    }
    public Triangle(Triangle ob) : base(ob) { Style = ob.Style; }
    public double Area() { return Width * Height / 2; }
    public void ShowStyle() {
        Console.WriteLine("Треугольник " + Style);
   }
}
```

Пример 12

```
class Shapes7 {
   static void Main() {
      Triangle t1 = new Triangle("прямоугольный", 8.0, 12.0);
      Triangle t2 = new Triangle(t1); // Сделать копию объекта t1
      Console.WriteLine("Сведения об объекте t1: ");
      t1.ShowStyle();          t1.ShowDim();
      Console.WriteLine("Площадь равна " + t1.Area());
      Console.WriteLine("Сведения об объекте t2: ");
      t2.ShowStyle();          t2.ShowDim();
      Console.WriteLine("Площадь равна " + t2.Area());
   }
}
```

Контрольные вопросы

1. Каков механизм наследования в языке C#?
2. Каким образом можно получить доступ из производного класса к закрытым полям базового класса?
3. Как реализуется вызов конструктора базового класса из производного?
4. В чем заключается механизм сокрытия членов базового класса и как можно обратиться к таким членам из производного класса?