

# Полиморфизм

# Полиморфизм



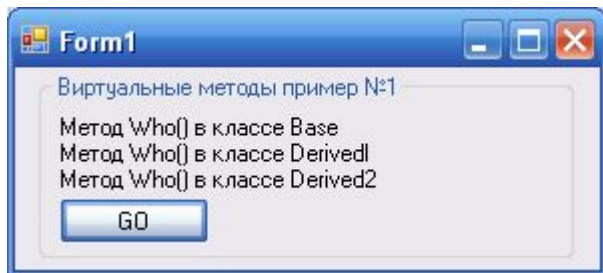
**Полиморфизм** – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

# Виртуальные методы

- Виртуальным называется такой метод, который объявляется как `virtual` в базовом классе. Виртуальный метод отличается тем, что он может быть переопределен в одном или нескольких производных классах. Следовательно, у каждого производного класса может быть свой вариант виртуального метода.
- Вариант выполняемого виртуального метода выбирается по типу объекта, а не по типу ссылки на этот объект. Так, если базовый класс содержит виртуальный метод и от него получены производные классы, то при обращении к разным типам объектов по ссылке на базовый класс выполняются разные варианты этого виртуального метода.
- Метод объявляется как виртуальный в базовом классе с помощью ключевого слова `virtual`, указываемого перед его именем. Когда же виртуальный метод переопределяется в производном классе, то для этого используется модификатор `override`. А сам процесс повторного определения виртуального метода в производном классе называется переопределением метода. При переопределении имя, возвращаемый тип и сигнатура переопределяющего метода должны быть точно такими же, как и у того виртуального метода, который переопределяется. Кроме того, виртуальный метод не может быть объявлен как `static` или `abstract`.
- Переопределение метода служит основанием для воплощения одного из самых эффективных в C# принципов: динамической диспетчеризации методов, которая представляет собой механизм разрешения вызова во время выполнения, а не компиляции. Значение динамической диспетчеризации

Пример. Создадим базовый класс виртуальным методом и два класса потомка, которые переопределяют данный метод.

```
class Base
{
    // Создать виртуальный метод в базовом классе.
    public virtual void Who(Label l)
    {
        l.Text="Метод Who() в классе Base";
    }
}
class Derived1 : Base
{
    // Переопределить метод Who() в производном классе.
    public override void Who(Label l)
    {
        l.Text="Метод Who() в классе Derived1";
    }
}
class Derived2 : Base
{
    // Вновь переопределить метод Who() в еще одном производном
классе.
    public override void Who(Label l)
    {
        l.Text="Метод Who() в классе Derived2";
    }
}
```



```
private void button1_Click(object sender, EventArgs e)
{
    Base b = new Base();
    Derived1 d1 = new Derived1();
    Derived2 d2 = new Derived2();
    b.Who(label1);
    d1.Who(label2);
    d2.Who(label3);
}
```

- Если в производном классе не предоставляется собственный вариант виртуального метода, то используется его вариант из базового класса.
- Если при наличии многоуровневой иерархии виртуальный метод не переопределяется в производном классе, то выполняется ближайший его вариант, обнаруживаемый вверх по иерархии.

# Что дает переопределение

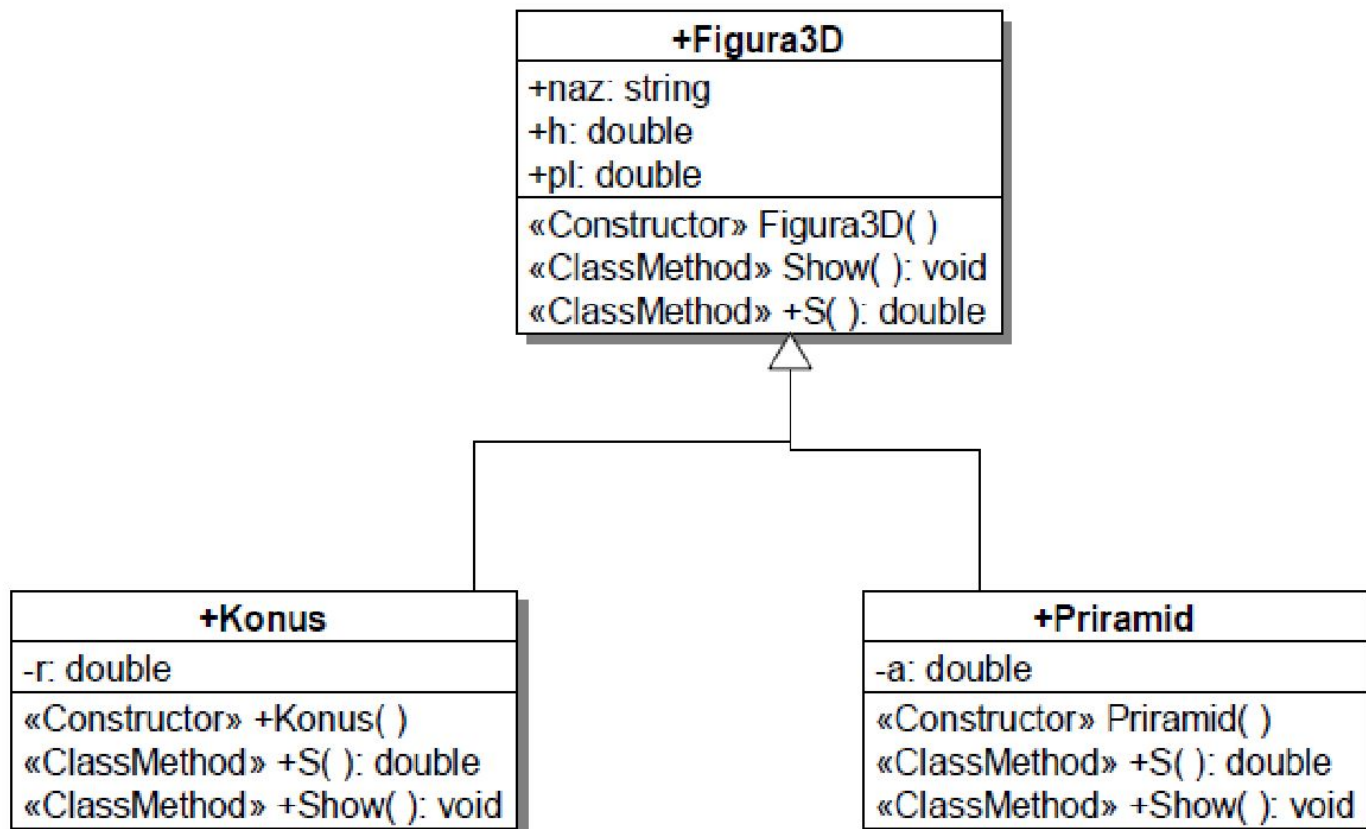
## методов

- Благодаря переопределению методов в C# поддерживается динамический полиморфизм.
- В объектно-ориентированном программировании полиморфизм играет очень важную роль, потому что он позволяет определить в общем классе методы, которые становятся общими для всех производных от него классов, а в производных классах — определить конкретную реализацию некоторых или же всех этих методов.
- Переопределение методов — это еще один способ воплотить в C# главный принцип полиморфизма: один интерфейс — множество методов.
- Удачное применение полиморфизма отчасти зависит от правильного понимания той особенности, что базовые и производные классы образуют иерархию, которая продвигается от меньшей к большей специализации. При надлежащем применении базовый класс предоставляет все необходимые элементы, которые могут использоваться в производном классе непосредственно. А с помощью виртуальных методов в базовом классе определяются те методы, которые могут быть самостоятельно реализованы в производном классе. Таким образом, сочетая наследование с виртуальными методами, можно определить в базовом классе общую форму методов, которые будут использоваться во всех его производных классах.

**Пример.** Составить программу с одним родительским классом и двумя потомками. Потомки должны содержать виртуальные функции. Создать виртуальную функцию выдачи результатов расчета методов на экран монитора с указанием названий и полей и их значений соответствующего объекта. Составить тестирующую программу с выдачей протокола на экран монитора. При этом создать объекты базового и производных типов, используя полиморфный контейнер - массив ссылок базового класса на объекты базового и производных классов (количество объектов  $\geq 5$ ).

Варианты	Родительский класс	Потомки	Полиморфные методы
1.	Трехмерная Фигура (поле название)	Конус(радиус $r$ , $h$ – высота) Правильная четырехугольная пирамида ( $a$ – сторона квадрата, $h$ -апофема, т.е. высота )	Расчет площади

# Диаграмма классов





# Описание классов

## Базовый класс для трехмерной фигуры

```
public class Figura3D
```

```
{  
    public string naz;  
    public double h;  
    public double p1;  
    public Figura3D()  
    {  
        int d = (int)DateTime.Now.Ticks;  
        Random rnd = new Random(d);  
        int r = rnd.Next(1, 10);  
        switch (r)  
        {  
            case 1: { naz = "Конус"; break; }  
            case 2: { naz = "Четырехугольная пирамида"; break; }  
            case 3: { naz = "Тор"; break; }  
            case 4: { naz = "Куб"; break; }  
            case 5: { naz = "Параллелепипед"; break; }  
            case 6: { naz = "Цилиндр"; break; }  
            case 7: { naz = "Сфера"; break; }  
            case 8: { naz = "Треугольная пирамида"; break; }  
            case 9: { naz = "Треугольная призма"; break; }  
            case 10: { naz = "Шестиугольная призма"; break; }  
        }  
        h = rnd.Next(1, 25);  
        p1 = rnd.Next(1, 25);  
    }  
    public Figura3D(string n)  
    {  
        naz = n;  
    }  
}
```

```
public virtual void Show(DataGridView dg)  
{  
    dg.Rows.Add("Название naz", naz);  
    dg.Rows.Add("Высота h", h.ToString());  
    dg.Rows.Add("Площадь S", p1.ToString());  
}  
  
public virtual double S()  
{  
    return 0;  
}
```

## Класс для конуса

```
class Konus : Figura3D
{
    private double r;
    public Konus()
    {
        int d = (int)DateTime.Now.Ticks;
        Random rnd=new Random(d);
        r=rnd.Next(1,10);
        h=rnd.Next(1,20);
        naz = "Конус";
    }
    public Konus(double rr, double hh)
    {
        r = rr;
        h = hh;
        naz = "Конус";
    }
    public override double S()
    {
        double s,l;
        l = Math.Sqrt(r * r + h * h);
        s = Math.PI * r * l;
        return s;
    }
    public override void Show(DataGridView dg)
    {
        pl = S();
        base.Show(dg);
        dg.Rows.Add("Радиус r", r.ToString());
    }
}
```

## Класс для пирамиды

```
class Piramid : Figura3D
{
    private double a;
    public Piramid()
    {
        int d = (int)DateTime.Now.Ticks;
        Random rnd=new Random(d);
        a=rnd.Next(1,15);
        h = rnd.Next(1, 25);
        naz = "Правильная четырехугольная пирамида ";
    }
    public Piramid(double aa,double hh)
    {
        a = aa;
        h = hh;
        naz = "Правильная четырехугольная пирамида ";
    }
    public override double S()
    {
        double s0,s4, s;
        s4 = 0.5 * a * h;
        s0 = a * a;
        s = 4 * s4 + s0;
        return s;
    }
    public override void Show(DataGridView dg)
    {
        pl = S();
        base.Show(dg);
        dg.Rows.Add("Сторона квадрата a", a.ToString());
    }
}
```

# Основная программа

```
private void button1_Click(object sender, EventArgs e)
{
    int n = Convert.ToInt16(textBox1.Text);
    //создаем контейнер ссылок на объекты базового класса
    Figura3D[] a = new Figura3D[n * 3];
    for (int i = 0; i < 3*n; i=i+3)
    {
        a[i] = new Figura3D();//создаем объект класса Figura3D
        a[i + 1] = new Konus(); //создаем объект класса Konus
        a[i + 2] = new Piramid(); //создаем объект класса Piramid
    }
    dataGridView1.Rows.Clear();
    //выводим поля объектов массива
    foreach (Figura3D el in a)
        el.Show(dataGridView1);
}
```

The screenshot shows a Windows application window titled "Form1". The window contains a data grid with 15 rows and 2 columns: "Поле" (Field) and "Значение" (Value). The grid displays properties for three different 3D objects: a cylinder, a cone, and a square pyramid. The first three rows correspond to the cylinder, the next three to the cone, and the last nine to the pyramid. The "Go" button is located at the bottom left, and a text box with the value "5" is at the bottom right.

Поле	Значение
Название naz	Цилиндр
Высота h	10
Площадь S	19
Название naz	Конус
Высота h	8
Площадь S	188,495559215388
Радиус r	6
Название naz	Правильная четырехугольная пирамида
Высота h	10
Площадь S	261
Сторона квадрата a	9
Название naz	Цилиндр
Высота h	10
Площадь S	19
Название naz	Конус
Высота h	8

введите количество объектов

Go

# Применение абстрактных

## классов

- Иногда требуется создать базовый класс, в котором определяется лишь самая общая форма для всех его производных классов, а наполнение ее деталями предоставляется каждому из этих классов. В таком классе определяется лишь характер методов, которые должны быть конкретно реализованы в производных классах, а не в самом базовом классе.
- Подобная ситуация разрешается двумя способами. Один из них, состоит в том, чтобы просто выдать предупреждающее сообщение. Второй состоит в использовании абстрактного метода.
- Абстрактный метод создается с помощью указываемого модификатора типа `abstract`. У абстрактного метода отсутствует тело, и поэтому он не реализуется в базовом классе. Это означает, что он должен быть переопределен в производном классе, поскольку его вариант из базового класса просто непригоден для использования.
- Абстрактный метод автоматически становится виртуальным и не требует указания модификатора `virtual`. В действительности совместное использование модификаторов `virtual` и `abstract` считается ошибкой.

Для определения абстрактного метода служит приведенная ниже общая форма.

`abstract тип имя(список_параметров);`

- У абстрактного метода отсутствует тело.
- Модификатор `abstract` может применяться только в методах экземпляра, но не в статических методах (`static`).
- Абстрактными могут быть также индексы и свойства.
- Класс, содержащий один или больше абстрактных методов, должен быть также объявлен как абстрактный, и для этого перед его объявлением `class` указывается модификатор `abstract`. А поскольку реализация абстрактного класса не определяется полностью, то у него не может быть объектов. Следовательно, попытка создать объект абстрактного класса с помощью оператора `new` приведет к ошибке во время компиляции.
- Когда производный класс наследует абстрактный класс, в нем должны быть реализованы все абстрактные методы базового класса. В противном случае производный класс должен быть также определен как `abstract`. Таким образом, атрибут `abstract` наследуется до тех пор, пока не будет достигнута полная реализация класса.

**Пример.** Составить программу с абстрактным родительским классом и двумя объектами - потомками. Для этого модифицировать задание 1. Составить тестирующую программу с выдачей протокола на экран монитора. В ней нужно реализовать циклический вывод параметров объектов, используя полиморфный контейнер - массив объектов базового класса (количество объектов  $\geq 5$ ).

Задание. Найти объект с максимальной площадью.

## Базовый класс для трехмерной

```
abstract class Figura3DA
{
    public double h;
    public double pl;
    public string naz;
    public abstract void Show(DataGridView dg);
    public abstract double S();
}
```

## Класс для

```
class KonusA : Figura3DA
{
    private double r;
    public KonusA(int xn, int xk)
    {
        Random rnd = new Random();
        r = rnd.Next(xn, xk);
        h = rnd.Next(xn+5, xk+10);
        naz = "Конус";
    }
    public KonusA(double rr, double hh)
    {
        r = rr;
        h = hh;
        naz = "Конус";
    }
    public override double S()
    {
        double s, l;
        l = Math.Sqrt(r * r + h * h);
        s = Math.PI * r * l;
        return s;
    }
    public override void Show(DataGridView dg)
    {
        pl = S();
        dg.Rows.Add("Название naz", naz);
        dg.Rows[dg.Rows.Count-1].DefaultCellStyle.ForeColor = Color.Red;
        dg.Rows.Add("Высота h", h.ToString());
        dg.Rows.Add("Радиус r", r.ToString());
        dg.Rows.Add("Площадь S", pl.ToString());
    }
}
```

# Класс для пирамиды

```
class PiramidA : Figura3DA
{
    private double a;
    public PiramidA(int xn,int xk)
    {
        Random rnd = new Random();
        a = rnd.Next(xn, xk);
        h = rnd.Next(xn+3, xk+5);
        naz = "Правильная четырехугольная пирамида ";
    }
    public PiramidA(double aa, double hh)
    {
        a = aa;
        h = hh;
        naz = "Правильная четырехугольная пирамида ";
    }
    public override double S()
    {
        double s0, s4, s;
        s4 = 0.5 * a * h;
        s0 = a * a;
        s = 4 * s4 + s0;
        return s;
    }
    public override void Show(DataGridView dg)
    {
        pl = S();
        dg.Rows.Add("Название naz", naz);
        dg.Rows[dg.Rows.Count - 1].DefaultCellStyle.ForeColor = Color.Green;
        dg.Rows.Add("Высота h", h.ToString());
        dg.Rows.Add("Сторона квадрата a", a.ToString());
        dg.Rows.Add("Площадь S", pl.ToString());
    }
}
```

# Основная программа

```
private void button2_Click(object sender, EventArgs e)
{
    int n = Convert.ToInt16(textBox2.Text);
    //создаем контейнер ссылок на объекты базового класса
    Figura3DA[] a = new Figura3DA[n * 2];
    double max = 0;
    string s="";
    for (int i = 0; i < 2*n; i=i+2)
    {
        a[i] = new KonusA(i,i+15); //создаем объект класса Konus
        a[i + 1] = new PiramidA(i+2,i+25); //создаем объект класса Piramid
    }
    dataGridView2.Rows.Clear();
    //выводим поля объектов массива
    foreach (Figura3DA el in a)
    { el.Show(dataGridView2);
      if (max< el.pl)
      {
          max = el.pl;
          s = el.naz;
      }
    }
    label3.Text = "Максимальная площадь " + max.ToString() + " " + s;
}
}
```



	Поле	Значение
▶	Название naz	Конус
	Высота h	7
	Радиус r	10
	Площадь S	383,480254480242
	Название naz	Правильная четырехугольная пирамида
	Высота h	8
	Сторона квадрата a	17
	Площадь S	561
	Название naz	Конус
	Высота h	9
	Радиус r	12
	Площадь S	565,486677646163
	Название naz	Правильная четырехугольная пирамида
	Высота h	10
	Сторона квадрата a	19
	Площадь S	741

введите количество объектов

Go

Максимальная площадь 1425 Правильная четырехугольная пирамида