

Лекция 10 ПРИНЦИП ПОЛИМОРФИЗМА

10.1 Понятие полиморфизма

Рассмотренное в предыдущей лекции понятие наследования позволяет использовать методы и данные базовых классов, но при этом различают два вида наследования – статическое и динамическое наследование (статическое и динамическое связывание методов).

Статическое наследование это такое наследование, все связи которого формируются во время компиляции программы и фактически определяются в самой структуре описания классов.

Динамическое наследование, и связанное с ним свойство полиморфизма, предполагают, что некоторые связи формируются в процессе выполнения программы.

Полиморфизм это многообразие форм реализации одноименных методов в цепочке наследуемых классов.

Реализация свойства полиморфизма осуществляется с помощью специальных виртуальных методов и, так называемых, абстрактных базовых классов.

Рассмотрим понятие абстрактных базовых классов.

Для получения преимуществ наследования классов разработчики ООП стали создавать базовые классы, включающих в себя все возможные методы обработки данных определенного множества объектов, но, при этом, базовые классы, как правило, не включали элементы данных.

Например, при создании базового класса «геометрические фигуры» можно включить методы нахождения площади или объема. Естественно, если производным классом является класс «точка» или класс «отрезок», то такие методы для этих объектов лишены смысла.

Базовые классы, для которых создание объектов невозможно или не имеет смысла, стали называть абстрактными базовыми классами. Абстрактные базовые классы служат только для порождения потомков. Как правило, в них задаются только наборы методов, которые каждый из потомков будет реализовывать по-своему. Подобные методы абстрактных базовых классов рассчитаны на несуществующие – виртуальные элементы данных (т.е. на элементы данных будущих классов в цепочке наследования).

Методы, рассчитаны на несуществующие, виртуальные элементы данных будущих классов в цепочке наследования, стали называть виртуальными методами.

Для обозначения виртуальных методов в языке C# используется специальное указание (специальный термин – `virtual`), означающее, что метод является виртуальным. Например:

```
virtual public double ploc() {return 0.0;}
```

Слово `virtual` в переводе с английского значит «фактический». Объявление метода виртуальным означает, что все ссылки на этот метод будут разрешаться по факту его вызова, то есть не на стадии компиляции, а

во время выполнения программы. Этот механизм называется поздним или динамическим связыванием методов.

Встретив виртуальные методы, компилятор создаст таблицу виртуальных методов (Virtual Method Table, VMT), в которую поместит названия виртуальных методов и адреса точек входа. Для каждого класса создается одна таблица виртуальных методов.

При этом во время выполнения программы в каждый создаваемый объект дополнительно включается указатель на созданную таблицу VMT.

Вызов виртуального метода выполняется так: из объекта берется адрес его таблицы VMT, из VMT выбирается адрес метода, а затем управление передается этому методу. Таким образом, при использовании виртуальных методов из всех одноименных методов иерархии всегда выбирается тот, который соответствует фактическому типу вызвавшего его объекта.

Если производный класс имеет свою реализацию одноименного виртуального метода, то в нем этот метод должен объявляться как замещающий или перекрывающий метод с атрибутом `override`. Например,

```
override public double ploc() { . . . }
```

Переопределять виртуальный метод в каждом из производных классов не обязательно. Если он выполняет устраивающие производный класс действия, то метод просто наследуется.

На этапе выполнения программы при обращении любого базового класса к замещенному методу в таблицу виртуальных методов помещается ссылка на соответствующий метод производного класса, который и работает так, как если бы он был изначальной частью родительского класса.

Замещающий виртуальный метод должен обладать таким же набором параметров, как и одноименный метод базового класса.

Принцип полиморфизма базируется на «перекрытии» виртуальных методов абстрактного базового класса замещающими методами. При этом каждый производный класс может иметь свою индивидуальную форму реализации наследуемых виртуальных или замещающих методов.

При этом свойство полиморфизма – это возможность для объектов разных классов, связанных наследованием, реагировать различным образом при обращении к одной и той же (по названию) виртуальной функции базового класса.

Полиморфизм, в переводе с греческого языка, означает «много форм», что в данном случае означает «один вызов — много методов».

При описании базовых классов рекомендуется определять в качестве виртуальных те методы, которые в производных классах должны реализовываться по-другому. Если во всех классах иерархии метод будет выполняться одинаково, его лучше определить как обычный метод.

10.2 Пример статического наследования методов

Разрабатываем цепочку наследуемых классов простейших геометрических фигур базовый, точка и круг.

В качестве базового возьмем класс, не имеющий полей и содержащий только виртуальную функцию вычисления площади и «чисто виртуальную функцию» печати.

Пример статического наследования – обычное создание объектов, присваивание полям данных (координатам точки или координатам центра круга и радиуса) некоторых случайных значений в диапазоне от 0 до 100 и печать этих значений.

Исходный код программы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public int x, y, ra;
        public abstract class baseGeo
        {
            public virtual double ploc() {return 0;}
            public abstract string printO();
        }

        public class tka : baseGeo
        {
            protected int x, y;
            public string ss;
            public tka()
            {
                Random rnd = new Random();
                x = rnd.Next(100);
                y = rnd.Next(100);
            }
            public int gettkax() { return x; }
            public int gettkay() { return y; }
            public void settka(int xx,int yy)
            {
                x=xx;
                y=yy;
            }

            override public string printO()
            {
                return "[" + x.ToString() + "," + y.ToString() + "]";
            }
        }
    }
}
```

```

public class kryg : tka
{
    public kryg()
    {
        Random rnd = new Random(10);
        r = rnd.Next(100);
    }
    public void setkryg(int ax,int ay,int rr)
    {
        settka(ax,ay);
        r=rr;
    }
    public void getkryg(out int ax, out int ay, out int rr)
    {
        ax = x;
        ay = y;
        rr = r;
    }
    public int getkrygr() { return r; }
    override public double ploc()    // перекрываем метод
                                    // базового класса
    {
        return 3.14*r*r;
    }
    override public string printO()    // перекрываем метод
                                    // класса tka
    {
        string ss = "";
        ss = "Круг с центром " + base.printO() + "\r\n";
        // наследуется функция печати класса tka
        ss = ss + " Радиусом = " + r.ToString() + "\r\n";
        return ss;
    }
    private int r;
}

public Form1()
{
    InitializeComponent();
}

private void button1_Click(object sender, EventArgs e)
{
    int i;
    double pl;
    string s;
    Random rnd = new Random(20);
    x = rnd.Next(100);
    y = rnd.Next(100);
    //ra = rnd.Next(100);

    // пример статического наследования методов

```

```

    tka a = new tka();
    a.settka(x, y);
    s = " Точка с координатами : " + a.printO() + "\r\n";
    textBox1.AppendText(s);
    s = " Площадь объекта = " + a.ploc().ToString() + "\r\n";
    textBox1.AppendText(s);
    kryg c = new kryg();
    x = c.gettkax();
    y = c.gettkay();
    ra = c.getkrygr();
    //c.setkryg(x, y, ra);
    textBox1.AppendText(c.printO());
    s = " Площадь объекта = " + c.ploc().ToString() + "\r\n";
    textBox1.AppendText(s);
    Invalidate();
}

private void Form1_Paint_1(object sender, PaintEventArgs e)
{
    Pen myPen = new Pen(Color.Red, 2);
    Graphics g = e.Graphics;
    g.DrawEllipse(myPen, x, y, ra, ra);
}
}
}

```

Работа программы:

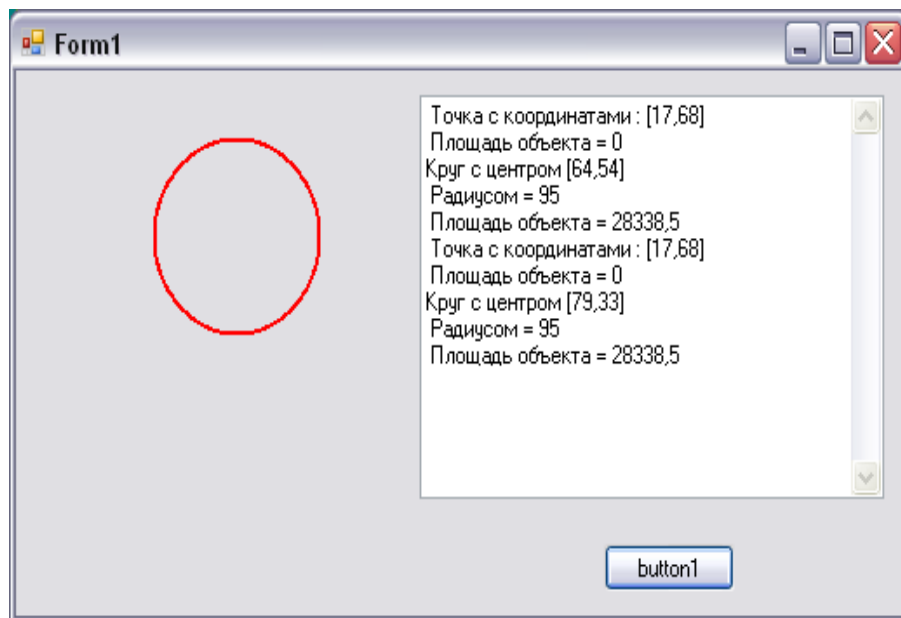


Рисунок 10.1 – Статическое наследование методов

В приведенном примере статического наследования методов присутствует механизм перекрытия методов, но связи между методами фактически определяются во время компиляции программы.

10.3 Пример динамического наследования методов

В качестве примера динамического наследования методов используем ту же цепочку наследуемых классов, но, во время работы программы, поместим 5 созданных объектов в стек.

Исходный код программы:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public int x, y, ra;
        public int[,] masi = new int[6,3];
        public Stack vstek = new Stack();
        public abstract class baseGeo
        {
            public virtual double ploc() { return 0; }
            public abstract string printO();
        }
        public class tka : baseGeo
        {
            protected int x, y;
            public tka()
            {
                Random rnd = new Random();
                x = (int)Math.Round(rnd.NextDouble() * 100);
                y = (int)Math.Round(rnd.NextDouble() * 100);
            }
            public int gettkax() { return x; }
            public int gettkay() { return y; }
            public void settka(int xx, int yy)
            {
                x = xx;
                y = yy;
            }
            override public string printO()
            {
                return "Точка [" + x.ToString() + "," + y.ToString() + "]";
            }
        }
        public class kryg : tka
        {
            public kryg()
```

```

{
    Random rnd = new Random(10);
    r = (int)Math.Round(rnd.NextDouble() * 100);
}
public void setkryg(int ax, int ay, int rr)
{
    settka(ax, ay);
    r = rr;
}
public void getkryg(out int ax, out int ay, out int rr)
{
    ax = x;
    ay = y;
    rr = r;
}
public int getkrygr() { return r; }
override public double ploc()           // перекрываем метод
                                         // базового класса
{
    return 3.14 * r * r;
}
override public string printO()         // перекрываем метод
                                         // класса tka
{
    string ss = "";
    ss = "Круг с центром: " + base.printO() + "\r\n";
    // наследуется функция печати класса tka
    ss = ss + " Радиусом = " + r.ToString() + "\r\n";
    return ss;
}
private int r;
}
static void vkl(Stack vst, baseGeo n)
{
    vst.Push(n);
}
public Form1()
{
    InitializeComponent();
}
private void button1_Click(object sender, EventArgs e)
{
    int i, k;
    double pl;
    string s;
    Stack vstek = new Stack();
    Random rnd = new Random(20);
    // пример динамического наследования методов
    for (i = 1; i <= 5; i++)
    {
        x = (int)Math.Round(rnd.NextDouble() * 100);
        y = (int)Math.Round(rnd.NextDouble() * 100);
        k = (int)Math.Round(rnd.NextDouble() * 100);
    }
}

```

```

if (k % 2 == 0) k = 0; else k = 1;
if (k == 0)
{
    tka a = new tka();
    a.settka(x, y);
    s = a.printO() + "\r\n";
    textBox1.AppendText(s);
    s = " Площадь объекта = " + a.ploc().ToString() + "\r\n";
    textBox1.AppendText(s);
    masi[i, 0] = a.gettkax();
    masi[i, 1] = a.gettkay();
    masi[i, 2] = 0;
    vkl(vstek, a);
}
else
{
    kryg c = new kryg();
    x = c.gettkax();
    y = c.gettkay();
    ra = c.getkrygr();
    textBox1.AppendText(c.printO());
    s = " Площадь объекта = " + c.ploc().ToString() + "\r\n";
    textBox1.AppendText(s);
    c.getkryg(out masi[i,0], out masi[i,1], out masi[i,2]);
    vkl(vstek, c);
}
}
s = "Печать содержимого стека: \r\n";
foreach (baseGeo T in vstek)
s = s + T.printO() + "\r\n";
textBox1.AppendText(s);
Invalidate();
}
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Pen myPen = new Pen(Color.Red, 2);
    Graphics g = e.Graphics;
    for (int i = 1; i <= 5; i++)
    {
        if (masi[i,2]==0)
        {
            x = masi[i, 0]; y = masi[i, 1];
            g.DrawEllipse(myPen, x, y, 2, 2);
        }
        else
        {
            x = masi[i, 0]; y = masi[i, 1]; ra = masi[i, 2];
            g.DrawEllipse(myPen, x, y, ra, ra);
        }
    }
}
}
}
}

```

Работа программы:

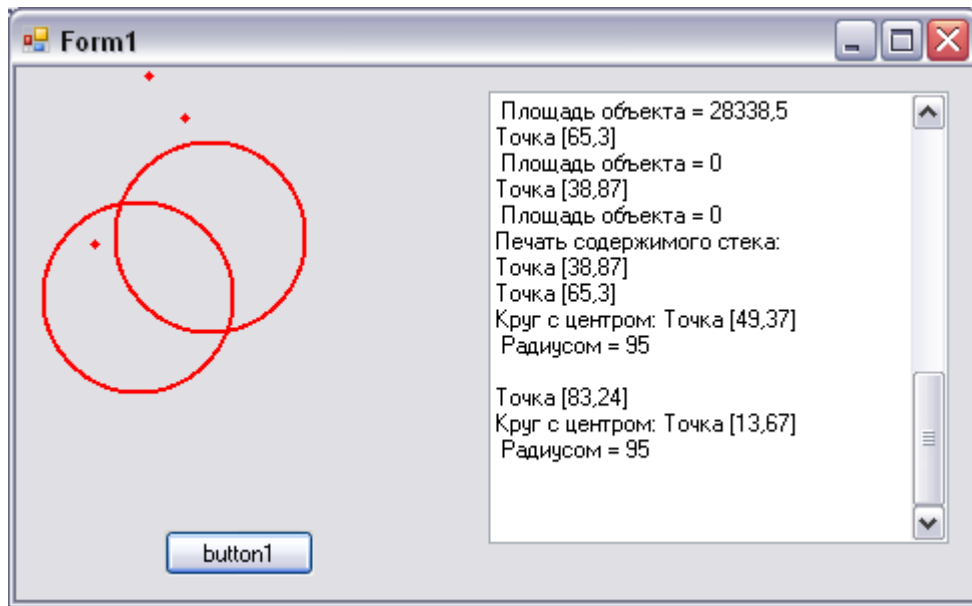


Рисунок 10.2 – Динамическое наследование методов

Печать содержимого элементов стека является примером полиморфизма (динамического наследования методов). Действительно, «статически» печать элемента стека рассчитана на печать элемента простейшей геометрической фигуры. Но в процессе работы программы (в этом суть динамического наследования методов) в стек помещаются объекты классов точка и круг.