Станка Матковић, Мијодраг Ђуришић

МАЛА ШКОЛА ОБЈЕКТНО ОРИЈЕНТИСАНОГ ПРОГРАМИРАЊА У ПРОГРАМСКОМ ЈЕЗИКУ C#

Четврти део

Наслеђивање, други део – апстрактне класе

У последњем примеру чланка из претходног броја могли смо закључити да је сваки учесник наставног процеса објекат неке од класа `Ucenik` или `Profesor`. Не постоји учесник наставног процеса који је објекат класе `Osoba`, па можемо рећи да класа `Osoba` само постоји да би објединила заједничке карактеристике и функционалности објеката изведених класа.

При процесу генерализације, издвајањем заједничких особина и функционалности неколико сродних класа, можемо креирати класу на вишем нивоу апстракције која се не материјализује кроз објекте (не креирамо њене објекте). Такву класу зовемо апстрактном класом. При њеној дефиницији неопходно је пре имена класе навести модификатор `abstract`.

```
abstract public class Osnovna
    {
// članovi klase
    }
```

Апстрактна класа је непотпуна. Неопходно је из ње изводити класе чије ћемо објекте креирати и дефинисати њихово понашање. У апстрактној класи наводимо заједничке функционалности свих изведених класа, које у изведеним класама можемо предефинисати. Наравно, свака од изведених класа може имати неке своје специфичне функционалности.

Врло често у класи која је добијена генерализацијом не можемо реализовати заједничку функционалност коју свака од изведених класа има али се у свакој од њих реализује на различит начин. Полазећи од облика попут круга, правоугаоника, квадрата, троугла генерализацијом долазимо до класе `Oblik`, а специјализацијом из те класе развијамо класе `Krug`, `Kvadrat`, `Pravougaonik`, `Trougao`. Сваки облик се може нацртати, и за сваки облик можемо израчунати обим и површину, али у класи `Oblik` не можемо реализовати те методе. Потребно је да класа `Oblik` има те фунционалности да би, коришћењем полиморфизма, сваком елементу

скупа различитих облика могли упутити захтев да их изврши. Сваки елемент ће на захтев (`Crtaj, Povrsina, Obim`) одговорити на себи својствен начин.

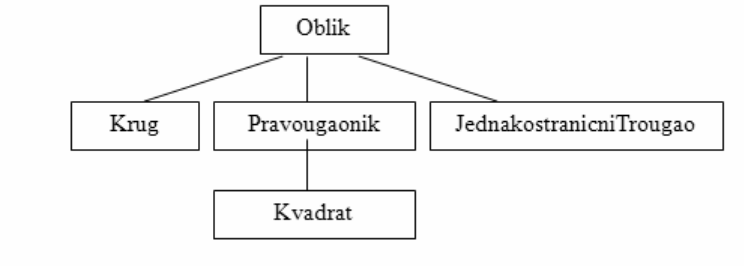
Метод апстрактне класе за који дајемо само декларацију а не и дефиницију, назива се апстрактни метод. Апстрактни метод пишемо навођењем модификатора `abstract` испред декларације метода коју завршавамо тачка зарезом. При декларацији апстрактног својства морамо навести да ли је то `get` и/или `set` својство.

```
abstract public povratniTip imeMetoda(listaParametara);
abstract public povratniTip imeSvojstva
    {
        get;
        set;
    }
```

У класама које су изведене из апстрактне класе при реализацији апстрактних чланова модификатор `abstract` замењујемо службеном речи `override`. Ако у изведеној класи нисмо реализовали све апстрактне чланове и та изведена класа је апстрактна па је потребно при њеној дефиницији навести модификатор `abstract`.

Пример 2. `Oblik`.

Наводимо реализацију апстрактне класе `Oblik` и из ње изведених класа `Krug`, `Pravougaonik`, `Kvadrat` и `JednakostranicniTrougao`. Посматрамо поједностављене облике, код квадрата и правоугаоника странице су паралелене координатним осама, а код једнакостраничног троугла једна страница је паралелна x-оси.



У основној класи `Oblik` дефинишемо заједничке (центар и боју облика) а у изведеним класама додатне атрибуте. Сви облици имају следеће функционалности: одређивање обима, одређивање површине, цртање, провера да ли облик садржи дату тачку и померање облика за дате вредности по x и y оси. Само последња од наведених функционалности реализује се на исти начин у свим класама, померањем центра облика за дате вредности. Зато тај метод реализујемо у основној класи и нема потребе да га у изведеним класама предефинишемо. Остале финкционалности (`crtaj, obim, povrsina, sadrzi`) не можемо реализовати у основној класи док не конкретизујемо облик. Зато су у класи `Oblik` ове функционалности апстрактне, а њихова реализација се налази у изведеним класама.

Напоменимо да је квадрат правоугаоник код којег дужина и ширина имају исте вредности, па у складу са тим класу `Kvadrat` добијамо специјализацијом из класе `Pravougaonik`.

```
public abstract class Oblik
    {
protected Color boja;
protected PointF centar;
public Oblik(Color boja, PointF centar)
        {
        this.boja = boja;
        this.centar = centar;
        }
public Oblik()
        {
        this.boja = Color.Black;
        this.centar = new PointF(0, 0);
        }
public void Pomeri(int dx, int dy)
        {
        centar.X += dx;
        centar.Y += dy;
        }
// apstraktna get svojstva za određivanje obima i površine
public abstract float Obim
        { get; }
public abstract float Povrsina
        { get; }
// apstraktni metod za crtanje oblika
public abstract void Crtaj(Graphics g);
// apstraktni metod za proveru da li oblik sadrži datu tačku
public abstract bool SadrziTacku(PointF A );
    }
public class Krug :  Oblik
    {
    float r;
    public Krug(Color boja, PointF centar, float r):base(boja, centar)
        {
        this.r = r;
        }
public Krug():base()
        {
        this.r = 10;
        }
public override void Crtaj(Graphics g)
        {
        g.FillEllipse(newSolidBrush(base.boja), base.centar.X - r, base.centar.Y - r,
2 * r, 2 * r);
        }
public override float Povrsina
        {
        get { return r * r * (float)Math.PI; }
        }
public override float Obim
        {
        get { return 2 * r * (float)Math.PI; }
        }
public override bool SadrziTacku(PointF M)
        {
```

```
        return Math.Sqrt((M.X - centar.X) * (M.X - centar.X) + (M.Y - centar.Y) *
(M.Y - centar.Y)) <= r;
        }
    }
public class JednakostranicniTrougao :  Oblik
    {
    float a;
    public JednakostranicniTrougao(Color boja, PointF centar,float a) :  base(boja,
centar)
        {
        this.a = a;
        }
public JednakostranicniTrougao() :  base()
        {
        this.a = 10;
        }
public override void Crtaj(Graphics g)
        {
        PointF[] teme = newPointF[3];
        float h=a*(float)Math.Sqrt(3) /2;
            teme[0] = new PointF(centar.X - a / 2, centar.Y + h / 3);
            teme[1] = new PointF(centar.X + a / 2, centar.Y + h / 3);
            teme[2] = new PointF(centar.X, centar.Y - 2*h / 3);
            g.FillPolygon(newSolidBrush(boja), teme);
        }
public override float Povrsina
        {
        get { return a * a * (float)Math.Sqrt(3) / 4; }
        }
public override float Obim
        {
        get { return 3 * a; }
        }
```

/*Proveru da li trougao sadrži tačku M vršimo tako što proverimo da li su tačka M i centar trougla sa iste strane svake od pravih određenih temenima trougla A i B, A i C, B i C. U takvom rešenju nismo koristili činjenicu da je jedna stranica trougla paralelna x osi, pa se ovo rešenje može lako uopštiti za proveru pripadnosti tačke proizvoljnom trouglu.*/

```
public override bool SadrziTacku(PointF M)
        {
        PointF A, B, C;
        float h = a * (float)Math.Sqrt(3) / 2;
            A = new PointF(centar.X - a / 2, centar.Y + h / 3);
            B = new PointF(centar.X + a / 2, centar.Y + h / 3);
            C = new PointF(centar.X, centar.Y - 2 * h / 3);
        return SaIsteStranePrave(A, B, M, centar) && SaIsteStranePrave(B, C, M, centar)
&& SaIsteStranePrave(A, C, M, centar);
        }
private static bool SaIsteStranePrave(PointF A, PointF B, PointF M, PointF N)
        {
        float a = B.Y - A.Y;
        float b = -(B.X - A.X);
        float c = -A.X * (B.Y - A.Y) + A.Y * (B.X - A.X);
        returnf(a, b, c, M) * f(a, b, c, N) >= 0;
        }
private static floatf(float a, float b, float c, PointF T)
        {
        return a * T.X + b * T.Y + c;
        }
    }
```

```
public class Pravougaonik :  Oblik
    {
    protectedfloat a, b;
    public Pravougaonik(Color boja, PointF centar, float a, float b) :  base(boja,
centar)
        {
        this.a = a;
        this.b = b;
        }
public Pravougaonik() :  base()
        {
        this.a = this.b = 10;
        }
public override void Crtaj(Graphics g)
        {
        g.FillRectangle(newSolidBrush(base.boja), centar.X-a/2,centar.Y-b/2,a,b);
        }
public override float Povrsina
        {
        get  return a * b;
        }
public override float Obim
        {
        get { return 2 * a + 2 * b; }
        }
public override bool SadrziTacku(PointF M)
        {
        return(M.X >= centar.X - a / 2)&&(M.X <= centar.X + a / 2)
        &&(M.Y >= centar.Y - b / 2)&&(M.Y <= centar.Y + b / 2);
        }
    }
public class Kvadrat :  Pravougaonik
    {
    public Kvadrat(Color boja, PointF centar, float a) :  base(boja,centar, a, a)
        { }
    public Kvadrat() :  base()
        { }
    }
```

Следи апликација у којој на случајан начин генеришемо n облика, приказујемо генерисане облике и обезбеђујемо померање превлачењем миша оних облика који садрже тачку на којој смо притиснули тастер миша.

Генерисане облике памтимо низом o чији су елементи класе `Oblik`. Класа `Oblik` је основна класа па елементи низа o могу бити објекти свих изведених класа из класе `Oblik`. Приликом цртања облика и провере да ли облик садржи дату тачку остварује се полиморфизам. За сваки елемент низа на исти начин позивамо метод за цртање `o[i].Crtaj(e.Graphics)`, при чему се позива предефинисани метод из одговарајуће класе у зависности од облика који елемент $o[i]$ садржи (`Krug, Pravouganik, JednakostranicniTrougao, Kvadrat`). Слично важи и при позиву метода за проверу да ли облик садржи дату тачку, `o[i].SadrziTacku(A)`.

```
Oblik[] o;
int n;
Random R = new Random();
bool[] pomeraj;
float x=-1, y=-1;
```


```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
    {
    for (int i = 0; i < n; i++)
    o[i].Crtaj(e.Graphics);
    }
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
    {
    n = (int)numericUpDown1.Value;
    o = newOblik[n];
    for (int i = 0; i < n; i++)
    {
int t = R.Next(1, 5);
Color boja = Color.FromArgb(R.Next(200, 256), R.Next(256), R.Next(256), R.Next(256));
PointFP=newPointF(R.Next(0,ClientRectangle.Width),R.Next(0,ClientRectangle.Height));
switch (t)
    {
    case 1:
        o[i] = newKrug(boja, P,R.Next(10,20)); break;
    case 2:
        o[i] = newJednakostranicniTrougao(boja, P, R.Next(20,50)); break;
    case 3:
        o[i] = newPravougaonik(boja,P, R.Next(10,50),R.Next(10,50)); break;
    case 4:
        o[i] = newKvadrat(boja, P, R.Next(10, 50)); break;
    }
    }
    pictureBox1.Refresh();
    // aktivira metodu pictureBox1_Paint
    }
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
    {
    x = e.X;
    y = e.Y;
    PointF A = new PointF(e.X, e.Y);
    pomeraj = newbool[n];
    for (int i = 0; i < n; i++)
    pomeraj[i] = o[i].SadrziTacku(A);
    }
private void picture Box1_MouseUp(object sender, MouseEventArgs e)
    {
```

```
    x = y = -1;
    }
private void picture Box1_MouseMove(object sender, MouseEventArgs e)
    {
    if (x != -1 && y != -1)
        {
        for (int i = 0; i < n; i++)
        if (pomeraj[i]) o[i].Pomeri(e.X - x, e.Y - y);
        x = e.X;
        y = e.Y;
        pictureBox1.Refresh();
        }
    }
```

Пример 3. `Funkcija`.

Посматрајмо математичке функције, дефинисане на скупу реалних бројева. Најпростије реалне функције су константе (функције облика $f(x) = c$, $c \in \mathbf{R}$) и променљиве ($f(x) = x$). Све остале градимо применом основних аритметичких операција или специјалних функција (тригонометријске, логаритамске, експоненцијалне ...) на претходно изграђене функције.

Генерализацијом можемо доћи до основне класе `Funkcija`. За сваку функцију можемо одредити вредност у задатој реалној тачки и нацртати график у одређеном интервалу. Одређивање вредности се разликује од функције до функције па метод `Vrednost(double x)` мора бити апстрактан као и сама класа. Поступак цртања графика је за све функције исти (спајамо тачке са координатама `(x, Vrednost(x))` за узастопне вредности x) па га реализујемо у класи `Funkcija` (метод `Nacrtaj`). У основној класи можемо реализовати и операторе за основне аритметичке операције јер се извршавају на исти начин, без обзира на то које функције се примењују.

Из класе `Funkcija`, специјализацијом, изводимо класе `Konstanta, Promenljiva, SlozenaFunkcija, Sinusna` ... У свакој од ових класа мора се предефинисати апстрактни метод `Vrednost`. Класе `SlozenaFunkcija` и `Sinusna` за атрибуте имају друге функције што је у складу са одговарајућим математичким функцијама (знамо да је параметар синусне функције произвољна реална функција а сложена функција представља резултат примене аритметичких операција на произвољне реалне функције).

```
abstract public class Funkcija
{
  abstract public double Vrednost(double x);
  public void Nacrtaj(Graphics g, PointF centar, double x0, double x1, float k)
//centar - (0,0), (x0,x1) - interval, k  broj pt u jedinici Dekartovog KS
  {
  for (doublex = x0; x<= x1; x+=0.001)
  g.DrawLine(Pens.Black, centar.X + (float)x * k, centar.Y  (float)Vrednost(x) * k,
centar.X + (float)(x+0.001) * k, centar.Y - (float)Vrednost(x+0.001) * k);
  }
  public static Funkcija operator +(Funkcija A, Funkcija B)
    {
    return new SlozenaFunkcija(A, B, '+');
    }
  public static Funkcija operator -(Funkcija A, Funkcija B)
```

```
    {
    return new SlozenaFunkcija(A, B, '-');
    }
public static Funkcija operator *(Funkcija A, Funkcija B)
    {
    return new SlozenaFunkcija(A, B, '*');
    }
public static Funkcija operator /(Funkcija A, Funkcija B)
    {
    return new SlozenaFunkcija(A, B, '/');
    }
public static Funkcij aoperator ^(Funkcija A, Funkcija B)
    {
    return new SlozenaFunkcija(A, B, '^');
    }
}
public class Konstanta :  Funkcija
    {
    double _c;
    public Konstanta(double c1)
    {
    _c = c1;
    }
public override double Vrednost(double x)
    {
    return _c;
    }
public override string ToString()
    {
    return _c.ToString();
    }
    }

public class Promenljiva :  Funkcija
    {
    public override double Vrednost(double x)
    {
    return x;
    }
public override string ToString()
    {
    return"x";
    }
    }

public class Sinusna :  Funkcija
    {
    Funkcija l;
    public Sinusna(Funkcija f)
    {
    l = f;
    }
public override double Vrednost(double x)
    {
    returnMath.Sin(l.Vrednost(x));
    }
public override string ToString()
    {
    return"sin("+l.ToString()+")";
    }
```

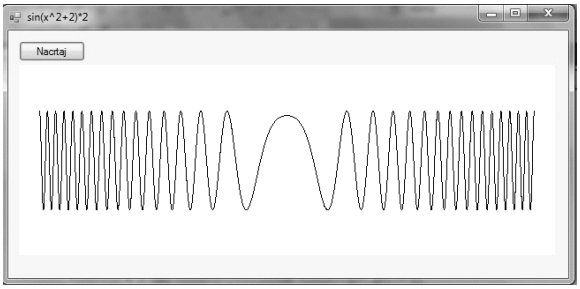
```
    }

public class SlozenaFunkcija :  Funkcija
    {
    Funkcija A;
    Funkcija B;
    Char operacija;
    public SlozenaFunkcija(Funkcija a, Funkcija b, Char c)
        {
        A = a;
        B = b;
        operacija = c;
        }
public override double Vrednost(double x)
    {
    switch (operacija)
    {
    case'+':  return A.Vrednost(x) + B.Vrednost(x);
    case'-':  return A.Vrednost(x) - B.Vrednost(x);
    case'*':  return A.Vrednost(x) * B.Vrednost(x);
    case'/':  return A.Vrednost(x) / B.Vrednost(x);
    case'^':  returnMath.Pow(A.Vrednost(x), B.Vrednost(x));
    }
    return 0;
    }
public override string ToString()
    {
    return A.ToString() + operacija + B.ToString();
    }
    }
```

Слично класи `Sinusna` можемо дефинисати и многе друге класе које представљају реалне функције чије је израчунавање реализовано у системској класи `Math` (тригонометријске, експоненцијалне, логаритамске, ...).

Наводимо и пример једноставне апликације која црта график функције $f(x) = \sin(x^2 + 2) \cdot 2$ на интервалу од -10 до 10.

```
private void btNacrtaj_Click(object sender, EventArgs e)
    {
    Graphics g=pictureBox1.CreateGraphics();
    Funkcija g1 = new Promenljiva(); //kreiranje promenljive x
    Funkcija g2 = new Konstanta(2); //kreiranje konstante 2
    Funkcija g3 = (g1 ^ g2) + g2; //kreiranje funkcije x^2+2
      Funkcija f = (newSinusna(g3)) * g2;// kreiranje funkcije sin(x^2+2)*2
      f.Nacrtaj(g, new PointF(pictureBox1.Width/2, pictureBox1.Height/2),-10,10,30);
      Text = f.ToString();
    }
```

Ово је само приказ основне идеје о класама помоћу којих можемо представити реалне функције. Реализацијом конструктора који за параметар имају стринг можемо значајно побољшати могућности овог система класа. Слично методу `Vrednost` можемо реализовати метод `Izvod` који би вратио функцију која представља први извод полазне функције. Остављамо читаоцима да сами усаврше овај софтвер у складу са својим жељама!



Закључак

Наслеђивање нам даје могућност коришћења већ дефинисане класе за израду нових класа које унапред добијају велики део функционалности, а имамо и могућност прилагођавања изведене класе својим потребама. На тај начин обезбеђујемо вишеструку употребу кода (исти код користимо у основној и изведеној класи). Захваљујући томе, програмирање је у претходном периоду остварило значајан напредак, јер су програмери усредсређени на решавање специфичних проблема а велики део пројекта, пре свега дизајн и структуру, наслеђују из класа груписаних у системске библиотеке или библиотека других програмера.

И у настави програмирања наслеђивање се може врло лепо искористити. Могуће је поступно надограђивати одређене пројекте кроз извођење све сложенијих класа. Такође, професор може креирати основну класу а ђацима препустити да реализују по групама разне изведене класе да би на крају заједно креирали врло употребљиве апликације.

Овим завршавамо „Малу школу ООП у C#“ на страницама вашег часописа. Надамо се да смо вас бар мало заинтересовали и да смо вам помогли да ове савремене методе програмирања приближите својим ђацима.

Математичка гимназија, Краљице Наталије 37, Београд