9 ОББ ПРИНЦИПТЕРІ

9.1 Инкапсуляция ұғымы

Объекті-бағытталған бағдарламалау технологиясы үш негізгі принциптерге негізделеді – инкапсуляция, мұрагерлік және полиморфизм.

Инкапсуляция принципі, яғни деректер мен оларды өңдеу әдістерінің бір құрылымға бірігуі класты ұйымдастыру негізін құрайды.

Формалды түрде инкапсуляция дегеніміз – деректерге қосымшадан тікелей қол жеткізуден қорғау мақсатында кластың өрістері мен әдістерінің бірігуі.

Объект өрістері қасиеттер немесе қол жеткізу ережелерінің жиыны – интерфейс арқылы қолданылуы тиіс. Объект өрістерін жасыру – олардың инкапсуляциясы («капсула» сөзінен) қасиеттер арқылы жүзеге асырылады.

Қасиеттер ұғымы алдынғы бөлімде толық қарастырылған, сондықтан қасиетті қосымшада қолдану мысалын ғана қарастырамыз.

Класс қасиеттерінің жұмысын көрсету үшін мысал ретінде treyg класын қолданамыз.

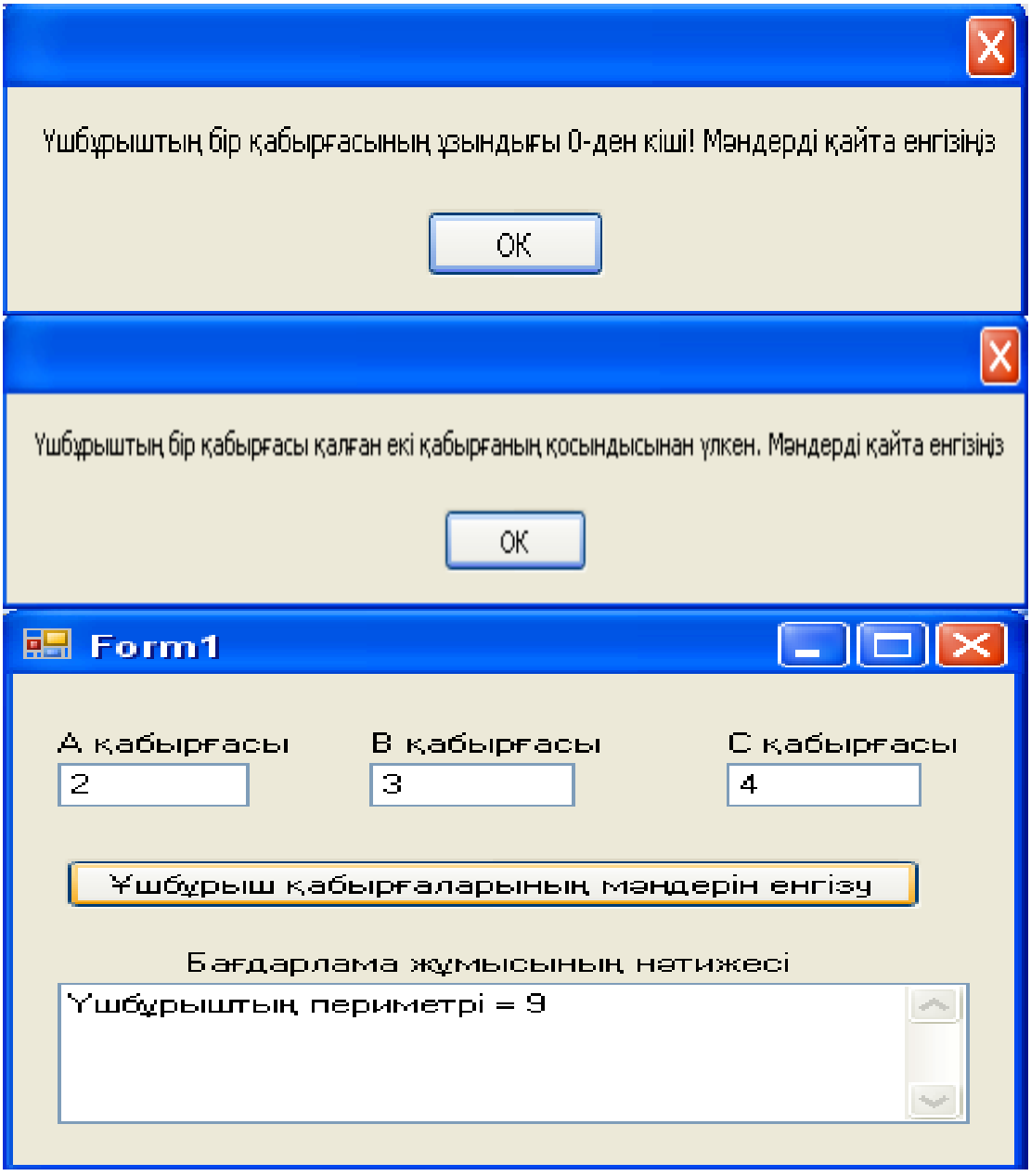
9.1-есеп. treyg класында үшбұрыш қабырғаларына қасиеттер арқылы кездейсоқ бүтін сандар меншіктеледі, сандар минус 5-тен 5-ке дейінгі аралықта болады. Қасиеттер мына шартты тексереді: енгізілген сандар 0-ден үлкен болуы керек. Үшбұрыш қабырғаларының мәндері енгізілгеннен кейін кластың арнайы әдісі мына шартты тексереді: кез келген екі қабырғаның қосындысы үшінші қабырғадан үлкен болуы керек. Қосымшаның жұмысына түсініктемелердің берілуі тиіс.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace WindowsApplication1
{
 public partial class Form1 : Form
 {
 public class treyg
 {
  private int a, b, c, p;
  public string ss;
  public treyg()
  {
  a = b = c = 0;
  }
  public int Aa
  {
  get { return a; }
  set { if (value > 0) a = value; else a = 0; }
  }
```

```
        public int Bb
        {
        get { return b; }
        set { if (value > 0) b = value; else b = 0; }
        }
        public int Cc
        {
        get { return c; }
        set { if (value > 0) c = value; else c = 0; }
        }
        public int Pp
        {
        get { return p; }
        }
        public void proverka()
        {
        if (a + b > c && a + c > b && b + c > a)
        {
         p = a + b + c;
         ss = ss + "Үшбұрыштың периметрі = " + p.ToString();
        }
        else
         MessageBox.Show("Үшбұрыштың бір қабырғасы қалған екі
қабырғаның қосындысынан үлкен. Мәндерді қайта енгізіңіз ");
        }
       }
       public Form1()
       {
        InitializeComponent();
       }
       private void button1_Click(object sender, EventArgs e)
       {
        Random rnd = new Random();
        int A = 1, B = 1, C = 1;
        bool ok = true;
        treyg t = new treyg();
        while (ok)
        {
        A = rnd.Next() % 11 - 5; t.Aa = A;
        B = rnd.Next() % 11 - 5; t.Bb = B;
        C = rnd.Next() % 11 - 5; t.Cc = C;
        textBox1.Text = Convert.ToString(t.Aa);
        textBox2.Text = Convert.ToString(t.Bb);
        textBox3.Text = Convert.ToString(t.Cc);
        if (A + B + C == t.Aa + t.Bb + t.Cc)
        {
         t.proverka();
         if (t.Pp != 0) ok = false;
        }
        else
         MessageBox.Show("Үшбұрыштың бір қабырғасының ұзындығы 0-
ден кіші! Мәндерді қайта енгізіңіз ");
        }
```

```
    textBox1.Text = Convert.ToString(t.Aa);
    textBox2.Text = Convert.ToString(t.Bb);
    textBox3.Text = Convert.ToString(t.Cc);
    textBox4.Text = t.ss;
   }
   }
}
```

Бағдарлама жұмысы 9.1-суретте көрсетілген.



9.1-сурет – Қосымшаның жұмыс терезесі

MessageBox.Show() терезесі шыққан кезде қосымшаның негізгі терезесінде үшбұрыш қабырғаларын сипаттайтын қасиеттердің мәндері шығады.

9.2 Мұрагерлік ұғымы

Мұрагерлік принципі объекті-бағытталған бағдарламалау тұжырымдамасында іргелі принцип болып табылады. Мұрагерліктің мақсаты – құрылып қойған кластарды қайталап қолдану.

Кейбір авторлар кластардың мұрагерлік идеясын түсіндіргенде жалпыдан жекеге қарай жалғасатын иерархиялық байланыстың мысалын келтіреді:

Жануарлар – мысық тектестер – жолбарыс.

Басқа авторлар кластардың мұрагерлік идеясын кіші объекттен үлкен объектке қарай жалғасатын иерархиялық байланыстың мысалы арқылы түсіндіреді:

Нүкте – кесінді – тіктөртбұрыш.

Материалды түсіндіруде авторлардың осы екі көзқарастары қолданылады.

«Кіші объекттен үлкен объектке қарай» тәсілі мұрагерлік идеясын түсіндіруге мүмкіндік береді – нүкте, одан кейін кесінді, т.б. Осы технологияны мұраланатын кластардың тізбегін «нөлден» дайындаған кезде қолдануға болады.

«Жалпыдан жекеге қарай» немесе «жалпыдан нақтыға қарай» екінші тәсілі бағдарламалауда құрылып қойған кластарды пайдалану арқылы қолданылады, мысалы, DELPHI-де VCL, VISUAL C++ ортасында MFC және басқа да стандартты кітапханалар, C# тілінде атаулар кеңістігі.

Деректері немесе әдістері мұраланатын класты базалық класс деп атайды.

Деректерді немесе әдістерді базалық кластан мұраға алатын класты туынды класс деп атайды.

Мұрагерлік туынды класқа өз қасиеттерін, деректерін, әдістерімен қатар базалық кластың қасиеттерін, деректерін, әдістерін қолдануға мүмкіндігін туғызады.

Мұрагерлік идеясын түсіндіру үшін келесі мысалды қарастырайық. Қосымшада құрылып қойған класқа қарағанда қосымша жаңа қасиеттер мен деректерге ие белгілі бір класты жазу керек болсын.

Оны жүзеге асырудың екі жолы бар.

Бірінші әдіс бойынша бір қосымшадағы класты жаңа қосымшадағы ға көшіріп, оған керекті өзгерістерді жазу керек.

Екінші әдіс бойынша алдынғы класты мұраға алатын (алдыңғы кластан туындайтын) екінші класты құру керек.

Бағдарламаны жазу бойынша қарастырылған екі әдістің ішінен ОББ тұрғысынан екінші әдіс қолайлы болып келеді.

Кластарды мұраға алу бағдарлама кодын айтарлықтай қысқартады және қосымшаны құру уақытын азайтып, оның сенімділігін жоғарлатады.

Мұрагерлік иерархиялық құрылымды құруға мүмкіндік береді, иерархиялық құрылымда туынды кластар базалық кластың өрістерін,

қасиетерін, әдістерін өз мүмкіндігіне алады және оларды толықтыра немесе өзгерте алады. Сонымен мұрагерлік кодты бірнеше рет қолдануға мүмкіндік береді. Базалық кластың кодын жазып, оны дұрыстағаннан кейін, осы класты мұралану арқылы оның кодын өзгертпей-ақ түрлі жағдайлар үшін қолдануға болады.

Иерархиялық құрылымның басына жақын орналасқан кластарда құрылымның төменгі жағындағы кластардың жалпы сипаттары біріккен. Иерархиялық құрылымның бойымен төмен жылжыған сайын кластардың айқын ерекшеліктері ұлғаяды.

C# тілінде кластардың иерархиялық тізбегінің базалық класы – System.Object.

Сонымен, мұрагерлік мына өзара байланысты мақсаттарда қолданылады:

– бағдарламада код үзінділерінің қайталануын болдырмау;

– бағдарламаның модификациясын жеңілдету;

– құрылған қосымшаны пайдаланып, жаңа қосымшаны құруды жеңілдету.

Сонымен қатар, мұрагерлікті пайдалану коды қол жетімді емес, бірақ өзгерістерді енгізуді қажет ететін объекттерді қолданудың бір ғана жолы болып келеді.

Кластарды мұралану бойынша жазу пішімінде класс әдеттегідей жарияланады, онда қос нүкте арқылы базалық кластың атауы жазылады:

```
    [ атрибуттар ] [ спецификаторлар ] class
класс_атауы [ : базалық класс ]
    { класс денесі}
```

Егер базалық кластың атауы көрсетілмесе, онда базалық класс болып System.Object класы есептеледі.

Егер бізде базалық класс бар болса, онда оны қос нүкте арқылы көрсету керек, мысалы:

```
public class otr : tka
{ класс денесі }
```

Мына мысалда кесінді класы нүкте класын мұраланады. Әлбетте, егер tka класының деректері privat қол жеткізу спецификаторы арқылы жабық болса, онда олар туынды класс үшін де жабық болады.

Кейбір базалық кластарда деректер protected қол жеткізу спецификаторынан кейін орналасады. Мысалы,

```
protected int x;
protected int y;
```

Қол жеткізу protected спецификаторының қажеттілігі мынада: туынды класс базалық кластың ашық элементтерін (public спецификаторы) қолдана алады.

Екінші жағынан туынды класс базалық кластың жабық элементтерін (private) тікелей қолдана алмайды, ол үшін туынды класс базалық кластың әдістерін қолдануы керек. Сондықтан кластың қорғалған элементтерін анықтайтын protected қол жеткізу спецификаторы пайда болды. Қорғалған элементтер жабық және ашық элементтердің ортасында аралық орынды алады. Егер элемент қорғалған болса, онда туынды класс объекттері оны ашық элемент сияқты қолдана алады. Қосымшанаң қалған бөлігіне қорғалған элементтер жабық болады. Ал егер қосымшада жабық элементтерді қолдану керек болса, онда оны тек кластың сәйкес әдістері арқылы ғана орындауға болады.

Класты мұралану кезіндегі негізгі мәселе – туынды кластың конструкторларын шақыру үшін объекттердің құрылу кезектілігін қарастыру.

Туынды класының әдістеріне базалық кластың деректері мен әдістерін мұралануға мүмкін болғандықтан, туынды класс объектісін дайындаған кезде базалық класта барлық деректері мен әдістері болуы тиіс.

Сондықтан туынды класс конструкторының жұмысы базалық класс конструкторын шақырудан басталады, оның жұмысы аяқталғаннан кейін туынды класс объектісі құрылады.

Туынды класс конструкторы мұраланатын базалық класс деректерінің элементтерін инициализациялауы тиіс.

Туынды класс деструкторын шақырған кезде бірінші болып туынды класс объектісі жойылуы керек, ал одан кейін – базалық класс объектісі.

Мұралану мәселесін қарастырған кезде мынаны ескеру керек: базалық кластың конструкторы мен деструкторы туынды класқа мұраға берілмейді.

9.2-есеп. Мұраланатын кластар тізбегін құру керек: нүкте-кесінді. Класс объекттерінің құрылу кезектілігін көрсететін мүмкіндікті қарастыру керек.

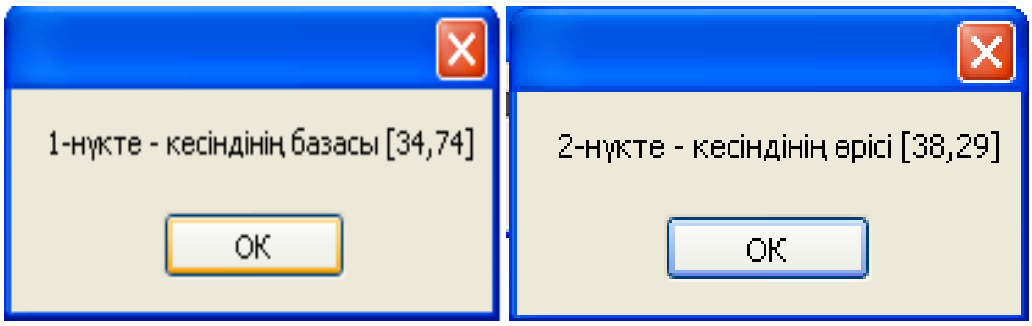
Form1.cs файлының коды:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace WindowsApplication1
{
 public partial class Form1 : Form
 {
 public int ax, ay, bx, by;
 public class tka
 {
  Random rnd = new Random();
  protected int x, y;
  public string ss;
  public tka()
  {
  x = rnd.Next(100);
  y = rnd.Next(100);
```

```
        }
        public void gettka(out int ax, out int ay)
        {
        ax = x; ay = y;
        }
        public string printtka()
        {
        return "[" + x.ToString() + "," + y.ToString() + "]";
        }
       }
       public class otr : tka
       {
        public tka b;
        public string s;
        public otr()
        {
        MessageBox.Show("1-нүкте - кесіндінің базасы [" +
x.ToString() + "," + y.ToString() + "]");
        s = "";
        }
        public void getotr(out int ax, out int ay)
        {
        int xx, xy;
        b.gettka(out xx, out xy);
        MessageBox.Show("2-нүкте - кесіндінің өрісі [" +
xx.ToString() + "," + xy.ToString() + "]");
        ax = xx; ay = xy;
        }
        public double dlina()
        {
        double dl;
        int k1, k2, k3, k4;
        k1 = x; k2 = y;
        b.gettka(out k3, out k4);
        dl = Math.Sqrt((k1 - k3) * (k1 - k3) + (k2 - k4) * (k2 -
k4));
        return dl;
        }
        public void printotr()
        {
        s = "Кесіндідегі нүкте координаты = " + b.printtka();
        s = s + "Кесіндінің базасындағы нүкте координаты = [" +
x.ToString() + "," + y.ToString() + "]";
        s = s + " Кесіндінің ұзындығы = " +
dlina().ToString("###.###");
        }
       }
       public Form1()
       {
        InitializeComponent();
       }
       private void button1_Click(object sender, EventArgs e)
       {
```

```
  string str;
  otr c = new otr();
  c.gettka(out bx, out by);
  tka bb = new tka();
  c.b = bb;
  c.getotr(out ax, out ay);
  c.printotr();
  str = c.s;
  textBox1.Text = str;
  Invalidate();
 }
 private void Form1_Paint(object sender, PaintEventArgs e)
 {
  Pen myPen = new Pen(Color.Red, 2);
  Graphics g = e.Graphics;
  g.DrawLine(myPen, ax, ay, bx, by);
 }
 }
}
```

Қосымшаның жұмысы 9.2-ші және 9.3-суреттерінде көрсетілген.



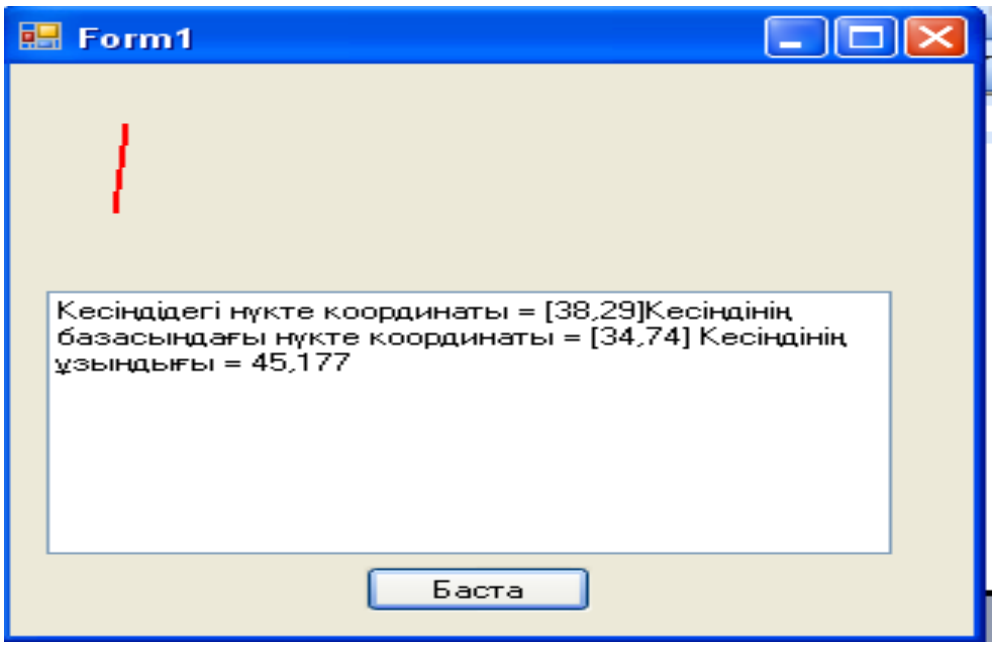
9.2-сурет – Кесінді ұштары координаталарының диалогтық терезелері

Кесінді объектісін құру кезінде базалық класс – нүкте объектісі бірінші болып құрылады (1-диалогтық терезе арқылы тексеру).

Одан кейін нүкте объектісін құраймыз және оны кесінді объектісінің өрісіне меншіктейміз (2-диалогтық терезе арқылы тексеру).

Кесінді ұзындығые есептейміз. Барлық нәтижелерді көпжолды мәтін редакторына шығарамыз.

X,Y координаттар жүйесіне кесіндіні саламыз (координаттар жүйесінің басы – форманың сол жақ жоғарғы бұрышы). Form1_Paint оқиға өңдеуішін құраймыз: форма үшін Properties терезесінде Paint оқиғасын таңдаймыз. Геометриялық фигуралар мен сызықтар үшін объекттерді дайындаймыз және кесінді координаталары бойынша сызықты саламыз.



9.3-сурет – Қосымшаның жұмыс терезесі

Оқиға өңдеуішін іске қосу (форма терезесін қайта салу) Invalidate() әдісі арқылы орындалады.

9.3 Өзін-өзі тексеру сұрақтары

1 Бір объект басқа объектке өз жағдайының өзгергені туралы қалай хабарлай алады?

2 Өз жағдайының өзгергені туралы басқа объектке хабарлайттын объект қалай аталады?

3 Кейбір оқиғалардың «бос» өңдеуіші қалай құрылады?

4 Формада button1 батырмасын екі рет шерткенде қандай оқиға өңдеуіші құрылады?

5 Кластарда операциялардың қайта анықтау не үшін қолданылады?

6 Мұрагерлік ұғымы?

7 Кластарды мұраланудың негізгі мақсаты неде?

8 Қасиеттерін, деректерін, әдістерін басқа класс мұраланған болса, онда ондай класс қалай аталады?

9 Базалық класс қасиеттерін, деректерін, әдістерін мұраға алатын класты қалай атайды?

10 ОББ неге негізделген?