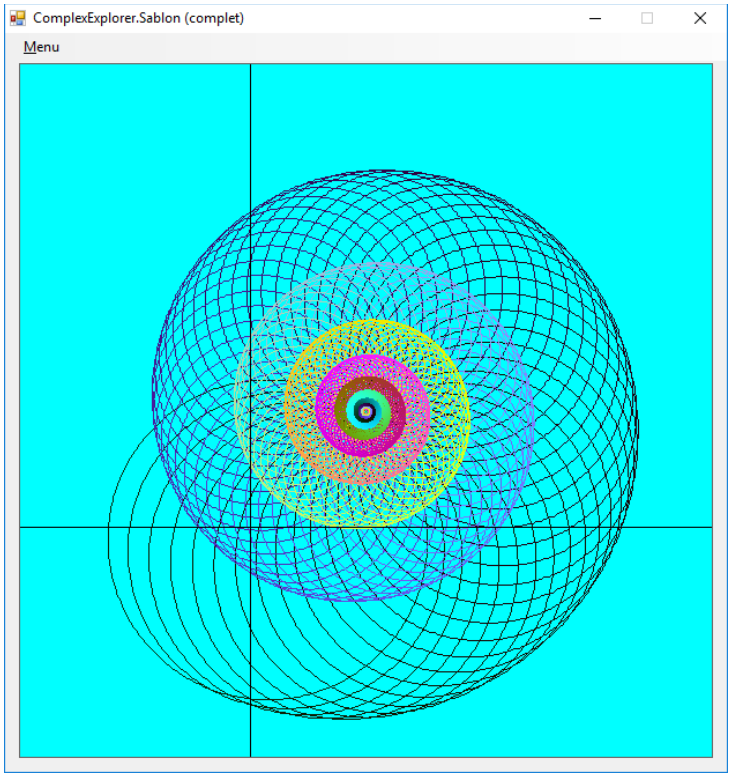
Cursul 2

Clase C# pentru grafică 2D

Proiectul ComplexExplorer descris mai jos a fost conceput ca un punct de plecare, ca un şablon iniţial în dezvoltarea de aplicaţii C# de grafică bidimensională cu ajutorul mediului de dezvoltare Microsoft Visual C# .

La rulare, programul afişează o formă cu dimensiuni fixe pe care se află o suprafaţă de desenare (de tip PictureBox), colorată iniţial în negru, şi un meniu cu cinci intrări (**Start**, **Stop**, **Save**, **About** şi **Exit**). Un clic pe **Start** declanşează trasarea desenului, **Stop** o opreşte, **Save** permite salvarea imaginii obţinute în format bitmap iar **Exit** închide programul. Submeniul **About** afişează căsuţa de dialog corespunzătoare.



Iată declaraţiile clasei formei principale (extrase din fişierul `ComplexForm.Designer.cs`), declaraţii scrise în mod automat de mediul de dezvoltare în etapa de proiectare:

```
partial class ComplexForm
    {
        ............
        protected System.Windows.Forms.PictureBox picBox;
        private System.Windows.Forms.MenuStrip mainMenu;
        private System.Windows.Forms.ToolStripMenuItem menuToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem startToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem stopToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem saveToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem aboutToolStripMenuItem;
        private System.Windows.Forms.SaveFileDialog mySaveFileDialog;
        private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
    }
```

Declaraţiile scrise de programator în cod sunt următoarele (în fişierul `ComplexForm.cs`):

```
public partial class ComplexForm : Form
    {
        public static AboutForm aboutForm = new AboutForm();
        private Graphics dc;
        private Bitmap bmp;
        private SolidBrush sdBrush;
        private Color[] paleta;
        public const int dimPaleta = 1024;
        public const int dimPicBmp = 600;
        public const int imin = 0, imax = dimPicBmp - 1, jmin = 0, jmax = dimPicBmp -
1;
        private double xmin, xmax, ymin, ymax;
        private double dxdi, dydj, didx, djdy;
        private bool doWork = false;
        private Color penColor=Color.White;
        private Color screenColor=Color.Black;
        ..............
   }
```

Bitmap-ul `bmp` este folosit pentru păstrarea în memorie a pixelilor desenaţi în program; pentru a deveni vizibil el este atribuit proprietăţii Image a picture-box-ului prin instrucţiunea

```
picBox.Image = bmp;
```

Atribuirea este făcută odată cu iniţializarea uneltelor grafice la încărcarea formei, în handler-ul de eveniment

```
private void ComplexForm_Load(object sender, EventArgs e)
        {
            bmp = new Bitmap(dimPicBmp, dimPicBmp, PixelFormat.Format24bppRgb);
            bmp.SetResolution(300F, 300F);
            picBox.Image = bmp;
            dc = Graphics.FromImage(bmp);
            sdBrush = new SolidBrush(screenColor);
        }
```

Pixelii bitmap-ului sunt setaţi cu ajutorul metodei `bmp.SetPixel(int i, int j, Color c)` iar reîmprospatarea imaginii pe ecran se obţine cu apelul `picBox.Invalidate();` efectuat de metoda `resetScreen()`.

In orice program de grafică 2D avem două sisteme de referinţă: cel al punctelor din plan, de coordonate (x,y), cu x şi y numere reale, şi cel al pixelilor de pe ecran, de coordonate (i, j), cu i şi j numere întregi. Pe ecran apar numai pixelii din dreptunghiul delimitat de `imin, imax, jmin, jmax,` care conţine imaginea din plan aflată în dreptunghiul delimitat de `xmin, xmax, ymin, ymax`. Transformările afine care suprapun cele două dreptunghiuri folosesc factorii de scalare `dxdi, dydj, didx, djdy,` calculaţi astfel:

```
dxdi = (xmax - xmin) / (imax - imin);
dydj = (ymax - ymin) / (jmax - jmin);
didx = (imax - imin) / (xmax - xmin);
djdy = (jmax - jmin) / (ymax - ymin);
```

Dreptunghiul pixelilor este fixat din start, la declararea câmpurilor corespunzătoare:

```
public const int imin=0, imax=dimPicBmp-1, jmin=0, jmax=dimPicBmp-1;
```

în timp ce dreptunghiul din planul xy poate fi setat de programator cu metoda

```
void setXminXmaxYminYmax(double xm, double XM, double ym, double YM);
```

Trecerea de la coordonatele (x, y) la (i, j) este dată de formulele:

```
i = (int)(imin + (x - xmin) * didx);
j = (int)(jmin + (y - ymin) * djdy);
```

iar trecerea inversă de formulele:

```
x = (double)(xmin+(i-imin)*dxdi);
y = (double)(ymin+(j-jmin)*dydj);
```

Culorile utilizate în aplicaţie pot fi iniţializate prin instrucţiuni de forma:

```
Color c1, c2, c3;
c1 = Color.Azure;
c2 = Color.FromArgb(128, 255, 17);
c3 = PenColor;
```

unde ultima variantă foloseşte proprietatea **PenColor** care returnează un obiect **Color** iniţializat la declararea clasei formei:

```
private Color penColor=Color.White;
```

sau prin utilizarea paletei de culori

```
private Color[] paleta;
```

care este setată în constructorul formei prin secvenţa:

```
this.paleta = new Color[dimPaleta];
for (int k = 0; k < dimPaleta; k++)
{
    double tcol = 56.123 + 2.0 * Math.PI * k / (double)dimPaleta;
    int rcol = (int)(128 + 128 * Math.Sin(tcol));
    int gcol = (int)(128 + 128 * Math.Sin(2 * tcol));
    int bcol = (int)(128 + 128 * Math.Cos(3 * tcol));
    paleta[k] = Color.FromArgb(rcol % 256, gcol % 256, bcol % 256);
}
```

Accesul public la culorile din paletă este permis numai prin intermediul metodei

```
public Color getColor(int k)     //Atentie: k trebuie sa fie pozitiv
{
     return paleta[k % dimPaleta];
}
```

Exemplu:

```
Color c1, c2;
c1 = paleta[453];          //in clasa ComplexForm
c2 = getColor(453);        //in afara
```

Câmpul `private bool doWork` al clasei `ComplexForm` are rolul de a indica dacă programul trebuie sau nu să stopeze desenarea. La selectarea itemului **Start** se setează `doWork` cu valoarea `true` şi este lansată în execuţie funcţia de desenare makeImage(). Apăsarea lui **Stop** sau **Exit** setează `doWork` la `false`. Desenarea este de obicei o operaţie de lungă durată care ţine procesorul ocupat un timp nepermis de lung şi din acest motiv în timpul desenării trebuie consultată, din când în când, valoarea variabilei `doWork` prin apelarea funcţiei

```
public bool resetScreen()
{
       picBox.Refresh();         //pentru re-pictarea ferestrei
      Application.DoEvents();    //pentru procesarea mesajelor
      return doWork;             //pentru semnalizarea opririi desenarii
}
```

In cazul în care rezultatul returnat are valoarea `false`, desenarea trebuie întreruptă.

Observăm că `resetScreen()` are un triplu rol:

1. *actualizarea imaginii*: invalidează imaginea obiectului **picBox** provocând astfel repictarea sa (astfel pixelii setaţi de noi în bitmap-ul din memorie apar pe ecran),

2. *procesarea mesajelor*: permite sistemului de operare să trateze evenimente apărute între timp şi să modifice, eventual, valoarea lui `doWork` (dacă utilizatorul a apăsat itemul **Stop,** de exemplu). Fără apelul metodei `DoEvents()` programul nostru ar bloca activitatea calculatorului până la sfârşitul desenării şi nu ar mai răspunde la evenimente.

3. *semnalizare*: informează funcţia apelantă, prin intermediul valorii lui `doWork`, dacă s-a apăsat **Stop** sau **Exit**, caz în care trebuie oprită desenarea.

Modul standard de lucru este următorul:

```
public override void makeImage()       //in clasa Sablon
{
      setXminXmaxYminYmax(-1.0, 1.0, -1.0, 1.0);
      ...
      for (int k = 0; k<1000000; k++)  //desenam o serie de figuri
      {
           col = getColor(k);
```

```
            for ( ...)                  //formam o figura intreaga
            {
                    ...
                    setPixel(x,y, col); // coloram  punct cu punct
                    ...
            }
            if (!resetScreen()) return; //arătăm ce am mai desenat,
                                  // vedem dacă trebuie să stopăm desenarea
      }
 }
```

Metodele grafice puse la dispoziţia programatorului au un caracter minimal (clasa Bitmap dispune numai de funcţia SetPixel, spre deosebire de clasa Graphics, special concepută pentru grafica 2D, dar care ar fi impus o abordare mai laborioasă). Mai precis, au fost implementate numai funcţiile:

```
protected void setPixel(int i, int j, Color c);
protected void setPixel(double x, double y, Color c);
protected void setPixel(Complex z, Color c);
protected void setLine(int i0, int j0, int i1, int j1, Color c);
protected void setLine(double x0, double y0, double x1, double y1, Color c);
protected void setLine(Complex z0, Complex z1, Color c);
```

In sfârsit, pentru trasarea rapidă a axelor de coordonate se poate apela metoda

```
protected void setAxis()
{
       setLine(xmin, 0.0, xmax, 0.0, penColor);
       setLine(0.0, ymin, 0.0, ymax, penColor);
}
```

Clasa **ComplexForm** are rolul de a fi clasă de bază pentru aplicaţii de grafică 2D. Pentru a facilita dezvoltarea acestora, clasa ComplexForm a fost prevăzută cu metoda virtuală

```
       public virtual void makeImage() {...}
```

care poate fi deci suprascrisă în clasele descendente. De fapt, o clasă derivată care execută un desen oarecare în plan se poate rezuma numai la suprascrierea acestei metode. In exemplul următor am preferat să dotăm clasa `Sablon` şi cu alte metode auxiliare, pentru a exemplifica o strategie generală de desenare a unei serii de figuri obţinute prin transformarea repetată a unei figuri geometrice iniţiale.

```
public class Sablon : ComplexForm
    {
        static Complex i = new Complex(0, 1);
        static Complex z0 = 5 + 5 * i;
        static Complex a = Complex.setRoTheta(0.99, Math.PI / 24);
```

```
Complex T(Complex z)
{
    return z0 + a * (z - z0);
}

Complex[] FiguraInitiala()    // un cerc centrat in origine
{
    int nrPuncte = 300;
    Complex[] rez = new Complex[nrPuncte];
    double ro = 7, delta = 2 * PI / nrPuncte;
    for (int k = 0; k < nrPuncte; k++)
    {
        rez[k] = Complex.setRoTheta(ro, k * delta);
    }
    return rez;
}
void Transforma(Complex[] fig)
{
    for (int k = 0; k < fig.Length; k++)
    {
        fig[k] = T(fig[k]);
    }
}
void Traseaza(Complex[] fig, Color col)
{
    for (int k = 1; k < fig.Length; k++)
    {
        setLine(fig[k - 1], fig[k], col);
    }

}
void Puncteaza(Complex[] fig, Color col)
{
    foreach (Complex z in fig)
    {
        setPixel(z, col);
    }
}
public override void makeImage()
{
    setXminXmaxYminYmax(-10, 20, -10, 20);
    PenColor = Color.Black;
    ScreenColor = Color.Aqua;
    setAxis();
    Complex[] fig = FiguraInitiala();
    for (int k = 1; k < 1000; k++)
    {
        Traseaza(fig, getColor(900 + 3 * k));
```

```
                //Puncteaza(fig, getColor(900 + 3 * k));
                Transforma(fig);
                if (!resetScreen()) return;
                //delaySec(0.1);
            }
            setAxis();
            resetScreen();
        }
    }
```

In acest exemplu figurile din plan sunt desenate cu ajutorul numerelor complexe, utilizând structura `Complex` definită în fisierul Complex.cs.

Prin *figură geometrică* noi vom înţelege întodeauna o mulţime finită de puncte din plan, memorată, cel mai adesea, sub forma unui tablou de numere complexe.

Să urmărim utilizarea acestor tablouri în metoda `makeImage()` a clasei `Sablon`. După ce, cu instrucţiunile

```
setXminXmaxYminYmax(-10, 20, -10, 20);
PenColor = Color.Black;
ScreenColor = Color.Aqua;
setAxis();
```

este stabilit dreptunghiul din planul xOy care va fi reprezentat pe ecran şi sunt trasate axele de coordonate, este încărcată în tabloul `fig` o figură iniţială, formată din afixele unui număr finit de puncte din plan. Mai precis, în ciclul **for** din funcţia `figuraInitiala()`

```
for (int k = 0; k < nrPuncte; k++)
{
    rez[k] = Complex.setRoTheta(ro, k * delta);
}
```

sunt puse în tablul returnat numerele complexe de pe cercul de rază `ro = 7`, centrat în origine, începând cu numărul de argument `theta = 0` şi continuând cu cele găsite prin parcurgerea cercului în sens trigonometric cu pasul `delta`.

Această figură iniţială este transformată apoi, în mod repetat de 500 de ori, prin transformarea

```
T(z)=z0+a*(z-z0);
```

transformare formată dintr-o rotaţie în jurul lui `z0 = 5+5*i` de unghi `arg(a)=PI/24`, compusă cu o omotetie de centru `z0` şi raport `|a|=0.99`.

Aplicarea transformării T asupra unei figuri memorate într-un tablou constă în parcurgerea tabloului și înlocuirea pe loc a fiecărui punct găsit cu imaginea sa prin transformarea dată:

```
void Transforma(Complex[] fig)
{
```

```
        for (int k = 0; k < fig.Length; k++)
        {
            fig[k] = T(fig[k]);
        }
    }
```

Desenarea unei figuri poate fi realizată cu metoda

```
    void Traseaza(Complex[] fig, Color col)
    {
        for (int k = 1; k < fig.Length; k++)
        {
            setLine(fig[k - 1], fig[k], col);
        }

    }
```

care trasează segmente colorate între punctele consecutive din listă, sau cu metoda

```
    void Puncteaza(Complex[] fig, Color col)
    {
        foreach (Complex z in fig)
        {
            setPixel(z, col);
        }
    }
```

care colorează doar punctele figurii.

Iniţial încărcăm în tabloul `fig` figura iniţială şi apoi repetăm de 1000 de ori următoarea secvenţă: modificăm elementele tabloului prin metoda de transformare și le desenăm în bitmap-ul `bmp`, apoi afişăm bitmap-ul pe ecran prin apelarea metodei `resetScreen()` (apel necesar şi din alte motive - vezi mai sus).

```
    public override void makeImage()
    {
        ......................
        Complex[] fig = FiguraInitiala();
        for (int k = 1; k < 1000; k++)
        {
            Transforma(fig);
            Traseaza(fig, getColor(900 + 3 * k));
            //Puncteaza(fig, getColor(900 + 3 * k));
            if (!resetScreen()) return;
            //delaySec(0.1);
        }
        setAxis();
        resetScreen();
    }
```

In final se obţine imaginea din instantaneul din prima pagină.

Anexe

1. Fişierul AboutForm.cs

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

namespace ComplexExplorer
{
    public  class AboutForm : Form
    {
        private string[] texte;
        private RichTextBox txtInfo;
        private Button btnOK;
        public AboutForm()
        {
            texte = new string[]{
            "Aplica\u021Bie \u00EEn Microsoft Visual C# pentru grafic\u0103 2D",
            "                    --- uz didactic ---",
            "Cursul \"Elemente de analiz\u0103 complex\u0103 \u0219i aplica\u021Bii\"",
            "Anul II, sec\u021Bia Matematic\u0103-informatic\u0103",
            "Facultatea de Matematic\u0103",
            "Universitatea \"Al. I. Cuza\" din Ia\u0219i", "Rom\u00E2nia"
            };


            this.Text = "About";
            this.StartPosition = FormStartPosition.WindowsDefaultLocation;
            this.FormBorderStyle = FormBorderStyle.FixedDialog;
            this.Height = 10 * this.Width / 16;
            this.ControlBox = false;


            txtInfo = new RichTextBox();
            txtInfo.Left = 15;
            txtInfo.Top = 18;
            txtInfo.Width = this.Width - 40;
            txtInfo.Height = 6 + this.Height / 2;
            txtInfo.Lines = texte;
            txtInfo.BackColor = Color.Gray;
            txtInfo.ForeColor = Color.WhiteSmoke;
            txtInfo.ReadOnly = true;
            txtInfo.Visible = true;
            this.Controls.Add(txtInfo);
```

```
            btnOK = new Button();
            btnOK.Text = "OK";
            btnOK.Left = this.ClientRectangle.Width - btnOK.Width - 20;
            btnOK.Top = this.ClientRectangle.Height - btnOK.Height - 8;

            btnOK.Visible = true;

            btnOK.Click += new EventHandler(btnOK_Click);
            this.Controls.Add(btnOK);
            btnOK.Select();
        }

        private void btnOK_Click(object sender, EventArgs eArgs)
        {

            this.Close();

        }
    }
}
```

2. Fişierul Complex.cs

```
using System;

namespace ComplexExplorer
{
    public struct Complex
    {
        private double x, y;

        public Complex(double x, double y)
        {
            this.x = x;
            this.y = y;
        }
        public override string ToString()
        {
            double eps = 1.0e-12;
            if (Math.Abs(y) < eps)              // z=real
                return string.Format("{0}", x);

            if (Math.Abs(x) < eps)              // z=imaginar
            {
                if (Math.Abs(y - 1) < eps)      //z=i
                    return "i";
                if (Math.Abs(y + 1) < eps)      //z=-i
                    return "-i";
                return string.Format("{0}i", y);
            }
            if (y > 0) return string.Format("{0}+{1}i", x, y);
```

```
    return string.Format("{0}{1}i", x, y);
}

public void show()
{
    Console.WriteLine(this);
}
public double Ro
{
    get
    {
        return Math.Sqrt(x * x + y * y);
    }
}
public double Ro2
{
    get
    {
        return x * x + y * y;
    }
}
public double Theta
{
    get
    {
        return Math.Atan2(y, x);
    }
}
public double Re
{
    get
    {
        return x;
    }
    set
    {
        x = value;
    }
}
public double Im
{
    get
    {
        return y;
    }
    set
    {
        y = value;
    }
}
```

```
        public Complex Conj
        {
            get
            {
                return new Complex(x, -y);
            }
        }

        public static Complex setRoTheta(double Ro, double theta)
        {
            return new Complex(Ro * Math.Cos(theta), Ro * Math.Sin(theta));
        }
        public static Complex setReIm(double x, double y)
        {
            return new Complex(x, y);
        }

        public static Complex operator +(Complex zst, Complex zdr)
        {
            return new Complex(zst.x + zdr.x, zst.y + zdr.y);
        }

        public static Complex operator +(Complex zst)
        {
            return new Complex(zst.x, zst.y);
        }

        public static Complex operator -(Complex zst, Complex zdr)
        {
            return new Complex(zst.x - zdr.x, zst.y - zdr.y);
        }
        public static Complex operator -(Complex zst)
        {
            return new Complex(-zst.x, -zst.y);
        }
        public static Complex operator *(Complex zst, Complex zdr)
        {
            return new Complex(zst.x * zdr.x - zst.y * zdr.y, zst.y * zdr.x + zst.x *
zdr.y);
        }
        public static Complex operator /(Complex zst, Complex zdr)
        {
            double r = zdr.Ro2;
            return new Complex((zst.x * zdr.x + zst.y * zdr.y) / r, (zst.y * zdr.x -
zst.x * zdr.y) / r);
        }

        public static implicit operator Complex(double x)
        {
```

```
            return new Complex(x, 0);
        }

        public static bool operator ==(Complex zst, Complex zdr)
        {
            return (zst - zdr).Ro2 < 1.0e-16;
        }
        public static bool operator !=(Complex zst, Complex zdr)
        {
            return (zst - zdr).Ro2 >= 1.0e-16;
        }

        public override bool Equals(object o)
        {
            if (o.GetType() != this.GetType()) return false;
            else return this == (Complex)o;
        }

        public override int GetHashCode()
        {
            return x.GetHashCode() + y.GetHashCode();
        }
    }
}
```

3. Fişierul ComplexForm.cs

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;
using System.Threading;

namespace ComplexExplorer
{
    public partial class ComplexForm : Form
    {
        public static AboutForm aboutForm = new AboutForm();
        private Graphics dc;
        private Bitmap bmp;
        private SolidBrush sdBrush;
        private Color[] paleta;
        public const int dimPaleta = 1024;
        public const int dimPicBmp = 600;
        public const int imin = 0, imax = dimPicBmp - 1, jmin = 0, jmax = dimPicBmp - 1;
        private double xmin, xmax, ymin, ymax;
        private double dxdi, dydj, didx, djdy;
        private bool doWork = false;
        private Color penColor=Color.White;
        private Color screenColor=Color.Black;
```

```
public ComplexForm()
{
    InitializeComponent();
    this.Height = dimPicBmp + 71;
    this.Width = dimPicBmp + 31;
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.ClientSize = new System.Drawing.Size(dimPicBmp + 25, dimPicBmp + 39);
    this.picBox.Location = new System.Drawing.Point(12, 26);
    this.picBox.Size = new System.Drawing.Size(dimPicBmp + 1, dimPicBmp + 1);
    this.picBox.BackColor = Color.Black;
    this.mainMenu.Location = new System.Drawing.Point(0, 0);
    this.mainMenu.Size = new System.Drawing.Size(dimPicBmp + 25, 24);
    this.KeyPreview = true;
    this.startToolStripMenuItem.Enabled = true;
    this.stopToolStripMenuItem.Enabled = false;
    this.saveToolStripMenuItem.Enabled = false;


    this.paleta = new Color[dimPaleta];
    for (int k = 0; k < dimPaleta; k++)
    {
        double tcol = 56.123 + 2.0 * Math.PI * k / (double)dimPaleta;
        int rcol = (int)(128 + 128 * Math.Sin(tcol));
        int gcol = (int)(128 + 128 * Math.Sin(2 * tcol));
        int bcol = (int)(128 + 128 * Math.Cos(3 * tcol));
        paleta[k] = Color.FromArgb(rcol % 256, gcol % 256, bcol % 256);
    }

    setXminXmaxYminYmax(0.0, 1.0, 0.0, 1.0);
    writeTitleBar("");
}

private void ComplexForm_Load(object sender, EventArgs e)
{
    bmp = new Bitmap(dimPicBmp, dimPicBmp, PixelFormat.Format24bppRgb);
    bmp.SetResolution(300F, 300F);
    picBox.Image = bmp;
    dc = Graphics.FromImage(bmp);
    sdBrush = new SolidBrush(screenColor);
}

private void ComplexForm_FormClosing(object sender, FormClosingEventArgs e)
{
     doWork = false;   //esential: ca sa stopam desenarea
}

public virtual void makeImage()     //trebuie suprascrisa
{
     setText("Metoda makeImage()","trebuie  suprascris\u0103!");
}

public void setText(params string[] mesaj)
{
    sdBrush.Color = penColor;
    using (Font font = new Font("Arial", 3.2f))
    {
```

```
            for (int k = 0; k < mesaj.Length; k++)
                dc.DrawString(mesaj[k], font, sdBrush, 0.0F, k*15.0F);
        }

    }


    public Color PenColor { get { return penColor; } set { penColor = value; } }

    public Color ScreenColor
    {
        get
        {
            return screenColor;
        }
        set
        {
            screenColor = value;
            initScreen();
            if (penColor.ToArgb() != value.ToArgb()) return;
            penColor = Color.FromArgb(value.A, 255 - value.R, 255 - value.G,
255 - value.B);
        }
    }



    public void initScreen()
    {
        sdBrush.Color = screenColor;
        dc.FillRectangle(sdBrush, 0, 0, dimPicBmp, dimPicBmp);
        picBox.Invalidate();
    }

    public bool resetScreen()
    {
        picBox.Refresh();        //pentru re-pictarea ferestrei
        Application.DoEvents();     //pentru procesarea mesajelor
        return doWork;              //pentru semnalizarea opririi desenarii
    }


    private void writeTitleBar(string msg)
    {
        this.Text = this.GetType().ToString();
        if (msg != "") this.Text += " (" + msg + ")";
    }

    private void startToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.startToolStripMenuItem.Enabled = false;
        this.stopToolStripMenuItem.Enabled = true;
        this.saveToolStripMenuItem.Enabled = false;
        initScreen();
        doWork = true;
        writeTitleBar("in lucru");
        makeImage();
        if (doWork) writeTitleBar("complet");
```

```
    else writeTitleBar("oprit");
    stopToolStripMenuItem_Click(null, null);
}

private void stopToolStripMenuItem_Click(object sender, EventArgs e)
{
    doWork = false;
    this.startToolStripMenuItem.Enabled = true;
    this.stopToolStripMenuItem.Enabled = false;
    this.saveToolStripMenuItem.Enabled = true;
}
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}
private void saveToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (bmp == null) return;
    mySaveFileDialog.Filter = "bitmap files (*.bmp)|*.bmp";

    if (mySaveFileDialog.ShowDialog() == DialogResult.OK
        && mySaveFileDialog.FileName.Length > 0)
    {

        bmp.Save(mySaveFileDialog.FileName, ImageFormat.Bmp);
    }
}
private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
{
    aboutForm.ShowDialog();
}

private void ComplexForm_KeyPress(object sender, KeyPressEventArgs e)
{                // tasta <space> lanseaza/opreste desenarea
    if (e.KeyChar != ' ') return;
    e.Handled = true;
    if (!doWork) startToolStripMenuItem_Click(null, null);
    else stopToolStripMenuItem_Click(null, null);

}
public void delaySec(double s)
{
    if (s <= 0 ) return;
    if(s>10) s=10;
    Thread.Sleep((int)(1000 * s));
}


public void setXminXmaxYminYmax(double xm, double XM, double ym, double YM)
{
    xmin = xm;
    xmax = (XM != xm ? XM : XM + 1.0e-10);
    ymin = ym;
    ymax = (YM != ym ? YM : YM + 1.0e-10);
    dxdi = (xmax - xmin) / (imax - imin);
    dydj = (ymax - ymin) / (jmax - jmin);
    didx = (imax - imin) / (xmax - xmin);
    djdy = (jmax - jmin) / (ymax - ymin);
```

```
        }
        public double Xmin { get { return xmin; } }
        public double Xmax { get { return xmax; } }
        public double Ymin { get { return ymin; } }
        public double Ymax { get { return ymax; } }

        public int getI(double x) { return (int)(imin + (x - xmin) * didx); }
        public int getJ(double y) { return (int)(jmin + (y - ymin) * djdy); }
        public double getX(int i) { return xmin + (i - imin) * dxdi; }
        public double getY(int j) { return ymin + (j - jmin) * dydj; }
        public Complex getZ(int i, int j) { return new Complex(xmin + (i - imin) * dxdi,
ymin + (j - jmin) * dydj); }

        public Color getColor(int k)     //Atentie: k trebuie sa fie pozitiv
        {
            return paleta[k % dimPaleta];
        }

        public void setPixel(int i, int j, Color c)
        {
            if (i < imin || imax < i || j < jmin || jmax < j) return;
            bmp.SetPixel(i, jmax - j, c);
        }


        public void setPixel(double x, double y, Color c)
        {
            setPixel((int)(imin + (x - xmin) * didx),
 (int)(jmin + (y - ymin) * djdy), c);
        }
        public void setPixel(Complex z, Color c)
        {
            setPixel((int)(imin + (z.Re - xmin) * didx),
 (int)(jmin + (z.Im - ymin) * djdy), c);

        }

        public void setLine(int i0, int j0, int i1, int j1, Color c)
        {
            int i, j, dir;
            double m;   //panta dreptei
            //linie verticala:
            if (i0 == i1)
            {
                if (j0 <= j1)
                {
                    for (j = j0; j <= j1; j++)
                        setPixel(i0, j, c);
                }
                else
                {
                    for (j = j1; j <= j0; j++)
                        setPixel(i0, j, c);
                }
                return;
            }
            //linie orizontala sau oblica:
```

```
            m = (double)(j1 - j0) / (double)(i1 - i0);
            if (-1 <= m && m <= 1)
            {
                dir = (i0 < i1 ? +1 : -1);
                i = i0;
                while (i != i1)
                {
                    setPixel(i, (int)Math.Round(j0 + m * (i - i0)), c);
                    i += dir;
                }
            }
            else     //m<-1 || m>1
            {
                dir = (j0 < j1 ? +1 : -1);
                j = j0;
                while (j != j1)
                {
                    setPixel((int)Math.Round(i0 + (j - j0) / m), j, c);
                    j += dir;
                }
            }
            setPixel(i1, j1, c);
            return;
        }

        public void setLine(double x0, double y0, double x1, double y1, Color c)
        {
            setLine((int)(imin + (x0 - xmin) * didx), (int)(jmin + (y0 - ymin) * djdy),
                (int)(imin + (x1 - xmin) * didx), (int)(jmin + (y1 - ymin) * djdy), c);
        }

        public void setLine(Complex z0, Complex z1, Color c)
        {
            setLine((int)(imin + (z0.Re - xmin) * didx),
 (int)(jmin + (z0.Im - ymin) * djdy),
                (int)(imin + (z1.Re - xmin) * didx),
(int)(jmin + (z1.Im - ymin) * djdy), c);
        }
        public void setAxis()
        {
            setLine(xmin, 0.0, xmax, 0.0, penColor);
            setLine(0.0, ymin, 0.0, ymax, penColor);
        }

    }

}
```

4. Fişierul ComplexForm.Designer.cs

```
namespace ComplexExplorer
{
    partial class ComplexForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; other-
wise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>


 private void InitializeComponent()
        {
            this.picBox = new System.Windows.Forms.PictureBox();
            this.mainMenu = new System.Windows.Forms.MenuStrip();
            this.menuToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.startToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.stopToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.saveToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.aboutToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.exitToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.mySaveFileDialog = new System.Windows.Forms.SaveFileDialog();
            ((System.ComponentModel.ISupportInitialize)(this.picBox)).BeginInit();
            this.mainMenu.SuspendLayout();
            this.SuspendLayout();
            //
            // picBox
            //
            this.picBox.BackColor = System.Drawing.Color.Black;
            this.picBox.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
            this.picBox.Location = new System.Drawing.Point(12, 26);
            this.picBox.Name = "picBox";
            this.picBox.Size = new System.Drawing.Size(600, 600);
            this.picBox.TabIndex = 0;
            this.picBox.TabStop = false;
```

```
            //
            // mainMenu
            //
            this.mainMenu.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.menuToolStripMenuItem});
            this.mainMenu.Location = new System.Drawing.Point(0, 0);
            this.mainMenu.Name = "mainMenu";
            this.mainMenu.Size = new System.Drawing.Size(624, 24);
            this.mainMenu.TabIndex = 1;
            this.mainMenu.Text = "menu1";
            //
            // menuToolStripMenuItem
            //
            this.menuToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.-
ToolStripItem[] {
            this.startToolStripMenuItem,
            this.stopToolStripMenuItem,
            this.saveToolStripMenuItem,
            this.aboutToolStripMenuItem,
            this.exitToolStripMenuItem});
            this.menuToolStripMenuItem.Name = "menuToolStripMenuItem";
            this.menuToolStripMenuItem.Size = new System.Drawing.Size(50, 20);
            this.menuToolStripMenuItem.Text = "&Menu";
            //
            // startToolStripMenuItem
            //
            this.startToolStripMenuItem.Name = "startToolStripMenuItem";
            this.startToolStripMenuItem.Size = new System.Drawing.Size(107, 22);
            this.startToolStripMenuItem.Text = "&Start";
            this.startToolStripMenuItem.Click += new System.EventHandler(this.startTool-
StripMenuItem_Click);
            //
            // stopToolStripMenuItem
            //
            this.stopToolStripMenuItem.Name = "stopToolStripMenuItem";
            this.stopToolStripMenuItem.Size = new System.Drawing.Size(107, 22);
            this.stopToolStripMenuItem.Text = "Sto&p";
            this.stopToolStripMenuItem.Click += new System.EventHandler(this.stopTool-
StripMenuItem_Click);
            //
            // saveToolStripMenuItem
            //
            this.saveToolStripMenuItem.Name = "saveToolStripMenuItem";
            this.saveToolStripMenuItem.Size = new System.Drawing.Size(107, 22);
            this.saveToolStripMenuItem.Text = "Sa&ve";
            this.saveToolStripMenuItem.Click += new System.EventHandler(this.saveTool-
StripMenuItem_Click);
            //
            // aboutToolStripMenuItem
            //
            this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
            this.aboutToolStripMenuItem.Size = new System.Drawing.Size(107, 22);
            this.aboutToolStripMenuItem.Text = "&About";
            this.aboutToolStripMenuItem.Click += new System.EventHandler(this.aboutTool-
StripMenuItem_Click);
            //
            // exitToolStripMenuItem
            //
```

```
            this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
            this.exitToolStripMenuItem.Size = new System.Drawing.Size(107, 22);
            this.exitToolStripMenuItem.Text = "E&xit";
            this.exitToolStripMenuItem.Click += new System.EventHandler(this.exitTool-
StripMenuItem_Click);
            //
            // ComplexForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(624, 638);
            this.Controls.Add(this.picBox);
            this.Controls.Add(this.mainMenu);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.MainMenuStrip = this.mainMenu;
            this.MaximizeBox = false;
            this.Name = "ComplexForm";
            this.Text = "No title";
            this.FormClosing += new System.Windows.Forms.FormClosingEventHandler(this.-
ComplexForm_FormClosing);
            this.Load += new System.EventHandler(this.ComplexForm_Load);
            this.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.Complex-
Form_KeyPress);
            ((System.ComponentModel.ISupportInitialize)(this.picBox)).EndInit();
            this.mainMenu.ResumeLayout(false);
            this.mainMenu.PerformLayout();
            this.ResumeLayout(false);
            this.PerformLayout();
        }
        #endregion
        protected System.Windows.Forms.PictureBox picBox;
        private System.Windows.Forms.MenuStrip mainMenu;
        private System.Windows.Forms.ToolStripMenuItem menuToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem startToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem stopToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem saveToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem aboutToolStripMenuItem;
        private System.Windows.Forms.SaveFileDialog mySaveFileDialog;
        private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
    }
}
```

5. Fişierul Program.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Threading;

namespace ComplexExplorer
{
    public class Sablon : ComplexForm
    {
        static Complex i = new Complex(0, 1);
        static Complex z0 = 5 + 5 * i;
        static Complex a = Complex.setRoTheta(0.99, Math.PI / 24);
```

```
Complex T(Complex z)
{
    return z0 + a * (z - z0);
}

Complex[] FiguraInitiala()    // un cerc centrat in origine
{
    int nrPuncte = 300;
    Complex[] rez = new Complex[nrPuncte];
    double ro = 7, delta = 2 * PI / nrPuncte;
    for (int k = 0; k < nrPuncte; k++)
    {
        rez[k] = Complex.setRoTheta(ro, k * delta);
    }
    return rez;
}
void Transforma(Complex[] fig)
{
    for (int k = 0; k < fig.Length; k++)
    {
        fig[k] = T(fig[k]);
    }
}
void Traseaza(Complex[] fig, Color col)
{
    for (int k = 1; k < fig.Length; k++)
    {
        setLine(fig[k - 1], fig[k], col);
    }

}
void Puncteaza(Complex[] fig, Color col)
{
    foreach (Complex z in fig)
    {
        setPixel(z, col);
    }
}
public override void makeImage()
{
    setXminXmaxYminYmax(-10, 20, -10, 20);
    PenColor = Color.Black;
    ScreenColor = Color.Aqua;
    setAxis();
    Complex[] fig = FiguraInitiala();
    for (int k = 1; k < 1000; k++)
    {
        Traseaza(fig, getColor(900 + 3 * k));
        //Puncteaza(fig, getColor(900 + 3 * k));
        Transforma(fig);
```

```
                if (!resetScreen()) return;
                //delaySec(0.1);
            }
            setAxis();
            resetScreen();
        }
    }
    static class Program
    {
        [STAThread]
        public static void Main()
        {
            Application.Run(new Sablon());
        }
    }
}
```