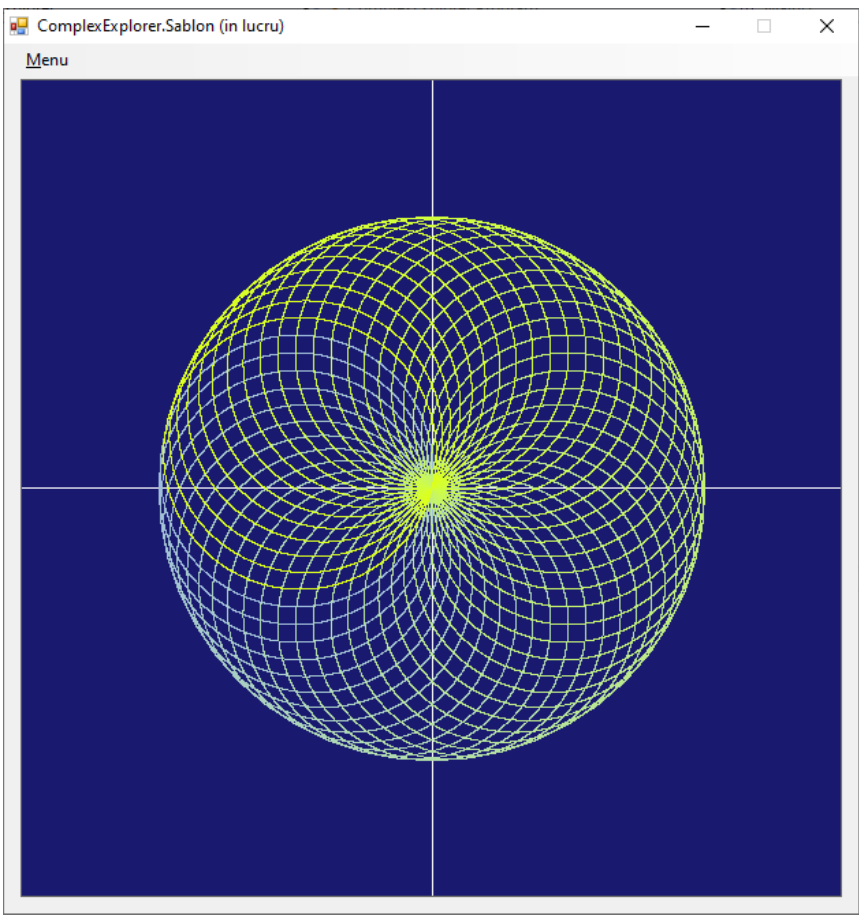
Cursul 2

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# .

Proiectul este din categoria Windows Forms Applications și, la rulare, afişează o formă cu dimensiuni fixe pe care se află o suprafaţă de desenare de tip `PictureBox` şi un meniu cu cinci intrări (`Start`, `Stop`, `Save`, `About` şi `Exit`). Un clic pe `Start` declanşează desenarea, `Stop` o opreşte, `Save` permite salvarea imaginii obţinute iar `Exit` închide programul. Submeniul `About` afişează căsuţa de dialog corespunzătoare.



Tot codul aplicației noastre este inclus în spațiul de nume

```
namespace ComplexExplorer
```

care se întinde pe mai multe fișiere sursă, fișiere analizate în continuare.

1. Fișieul `Program.cs`. Aici este definită clasa `Program` , cea care conține metoda `Main()`, punctul de start al programului:

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

namespace ComplexExplorer
{
    static class Program
    {
        [STAThread]
        public static void Main()
        {
            Application.Run(new Sablon());
            //Application.Run(new RandomList());
            //Application.Run(new SistemDinamic());
            //Application.Run(new CurbeLissajous());
            //Application.Run(new Pixeli());
        }
    }
}
```

După cum se observă, în exemplul de mai sus metoda `Main()` constă doar în apelul metodei statice `Run()` a clasei `Application`, care pune în funcție forma primită ca argument, în cazul nostru un obiect de tip `Sablon`, nou creat prin apelul constructorului clasei cu operatorul **`new`**.

2. Fișieul `MyToolbox.cs`. Acest fișier a fost adăugat la proiect special pentru a fi locul unde utilizatorul își poate scrie codul claselor definite de el. În exemplul analizat, aici găsim definiția clasei `Sablon` :

```
namespace ComplexExplorer{

public class Sablon : ComplexForm
    {
        Complex[] FiguraInitiala(double r)  // un cerc care trece prin origine
        {
            int nrPuncte = 5000;
            Complex[] rez = new Complex[nrPuncte + 1];
            double delta = 2 * Math.PI / nrPuncte;
            for (int k = 0; k <= nrPuncte; k++)
```

```
            {
                rez[k] = r + Complex.setRoTheta(r, k * delta);
            }
            return rez;
        }

        public override void makeImage()
        {
            double lat = 2;
            setXminXmaxYminYmax(-lat, lat, -lat, lat);
            setAxis();
            Complex[] fig = FiguraInitiala(lat/3);
            Complex rotor = Complex.setRoTheta(1, Math.PI / 24);
            for (int t = 0; t < 10000; t++)
            {
                //initScreen();
                Color col = getColor(2*t);
                for (int k = 0; k < fig.Length; k++)
                {
                    setPixel(fig[k], col);
                    fig[k] *= rotor;
                }
                if (!resetScreen()) return;

            }
            setAxis();
            resetScreen();
        }
    }
}
```

Prima linie de cod

```
public class Sablon : ComplexForm
```

arată că `Sablon` este o clasa derivată direct din clasa `ComplexForm`, prin urmare moştenește toate metodele acestei clase special create pentru realizarea de desene pixel cu pixel. La rândul ei, clasa `ComplexForm` este derivată din `System.Windows.Forms.Form`, prin urmare un obiect de tip `Sablon` este o formă Windows.

Codul clasei `Sablon` constă numai în definirea unei metode noi, `FiguraInitiala()`, şi în suprascrierea metodei `makeImage()`, metodă moștenită de la clasa bază. Definiția inițială a metodei moștenite o găsim în fișierul `ComplexForm.cs` :

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

Proiectul ComplexExplorer a fost conceput în acest fel pentru a ușura la maxim munca utilizatorului, acesta are numai două lucruri de facut: să își definească în fișierul `MyToolbox.cs` propria clasă derivată din `ComplexForm` în care să suprascriere metoda `makeImage()`

```
public class MyForm : ComplexForm
{
        ...
        public override void makeImage(){
                ...
        }
}
```

și apoi, în fișierul `Program.cs`, să îsi pună forma în funcțiune cu un apel la `Application.Run()` în interiorul metodei `Main()`

```
static class Program
{
        [STAThread]
        public static void Main()
        {
                Application.Run(new MyForm());
        }
}
```

Revenim la exemplul dat și analizăm codul clasei `Sablon`. Metoda

```
 Complex[] FiguraInitiala(double r)  // un cerc care trece prin origine
 {
     int nrPuncte = 5000;
     Complex[] rez = new Complex[nrPuncte + 1];
     double delta = 2 * Math.PI / nrPuncte;
     for (int k = 0; k <= nrPuncte; k++)
     {
         rez[k] = r + Complex.setRoTheta(r, k * delta);
     }
     return rez;
 }
```

primește ca parametru de intrare un număr `r` de tip `double` și returnează un tablou de numere complexe, și anume tabloul `rez`[1] declarat și inițializat astfel:

```
Complex[] rez = new Complex[nrPuncte + 1];
```

Aici prin "numere complexe" înțelegem obiecte de tip `Complex` definite de structura `struct Complex` descrisă în fișierul `Complex.cs`.

1 Denumirea rez este o prescurtare de la "rezultatul metodei".

In ciclul `for (int k = 0; k <= nrPuncte; k++)` sunt încărcate în tabloul `rez` un număr de 5001 puncte de pe cercul de rază `r` cu centrul în punctul de coordonate `(r,0)`, utilizând operațiile cu numere complexe.

În metoda `makeImage()`

```
public override void makeImage()
 {
     double lat = 2;
     setXminXmaxYminYmax(-lat, lat, -lat, lat);
     setAxis();
     Complex[] fig = FiguraInitiala(lat/3);
     Complex rotor = Complex.setRoTheta(1, Math.PI / 24);
     for (int t = 0; t < 10000; t++)
     {
         //initScreen();
         Color col = getColor(2*t);
         for (int k = 0; k < fig.Length; k++)
         {
             setPixel(fig[k], col);
             fig[k] *= rotor;
         }
         if (!resetScreen()) return;

     }
     setAxis();
     resetScreen();
 }
```

fixăm cu `setXminXmaxYminYmax(-lat, lat, -lat, lat)` dreptunghiul cadru al desenului nostru, aici pătratul `[-2,2] x [-2,2]` din plan, și apoi încărcăm în tabloul `Complex[] fig` cercul returnat de apelul `FiguraInitiala(lat/3)`. În continuare, în ciclul

```
for (int t = 0; t < 10000; t++)  {...}
```

rotim pas cu pas acest cerc în jurul originii, la fiecare pas unghiul de rotație fiind argumentul numărului complex `rotor`, adică `Math.PI / 24` radiani. Noua poziție a figurii o desenăm cu o altă culoare, obținută cu apelul metodei `getColor()` a clasei `ComplexExplorer`. Instrucțiunea

```
 setPixel(fig[k], col);
```

colorează cu culoarea `col` pixelul corespunzător numărului complex `fig[k]`, utilizând metoda

```
public void setPixel(Complex z, Color c)
```

a clasei `ComplexExplorer.`

Instrucțiunea

```
if (!resetScreen()) return;
```

are un rol triplu: unu, să împrospăteze (cu *refresh*) imaginea afișată pe monitor, ca să apară pe ecran pixelii nou colorați în ciclul `for (int k = 0; k < fig.Length; k++)` de deasupra, doi: să întrerupă puțin execuția metodei `makeImage()`, cât să fie procesate mesajele primite de aplicație de la sistemul de operare și trei, să încheie execuția metodei `makeImage()` cu `return` dacă aplicația a primit mesajul "a fost selectat itemul Stop". Definiția metodei se găsește în fișierul `ComplexForm.cs`:

```
 public bool resetScreen()
```

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

Câmpul privat `bool isWorking` este folosit pe post de semafor, itemul `Start` îl setează la `bool` iar `Stop` îl de-setează la `false`.

În ultimile două linii de cod retrasăm axele de coordonate, cu `setAxis()`, și apelăm din nou metoda `resetScreen()`, de data aceasta numai pentru refreșul imaginii.

Metoda `resetScreen()` nu trebuie confundată cu `initScreen()` care are o denumire asemănătoare dar o cu totul altă acțiune: șterge vechea imagine înlocuind-o cu un fundal de o culoare uniformă, dată de proprietatea publică `ScreenColor`:

```
        public void initScreen()
        {
            myBrush.Color = ScreenColor;
            myDC.FillRectangle(myBrush, 0, 0, latBmp, latBmp);
            picBox.Invalidate();
        }
```

Reținem: metoda `setPixel()` nu are un efect vizibil pe ecran, ea scrie într-un bitmap din memorie, pentru ca acesta să reapară pe ecran apelăm `resetScreen()` iar ca să îl ștergem folosim `initScreen()`.

3. Fișierul `ComplexForm.Designer.cs`. Acest fișier conține codul scris în mod automat de programul designer al mediul de dezvoltare în etapa de proiectare vizuală și conține o parte din definiția clasei ComplexForm.

Acum suntem interesați numai de declarațiile câmpurilor

```
namespace ComplexExplorer
{
    partial class ComplexForm
    {
        ....................................................................
        private 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;
    }
}
```

dintre care îl distingem pe `picBox`, o componentă de tip `PictureBox` utilizată drept cadru de afișare pe ecran a unui bitmap atribuit proprietății `picBox.Image`.

În acest fişier mai găsim şi codul metodei `InitializeComponent()`, metodă apelată de constructorul clasei, şi care cuprinde toate setările proprietăților formei efectuate în etapa de design.

4. Fişierul `ComplexForm.cs`. Aici găsim încă o serie de câmpuri ale clasei `ComplexForm`, clasă derivată după cum se vede din `System.Windows.Forms.Form`

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

namespace ComplexExplorer
{
    public partial class ComplexForm : Form
    {
        private static AboutForm aboutForm = new AboutForm();
        private Graphics myDC;  // my device context
        private Bitmap myBmp;   // my bitmap
        private SolidBrush myBrush;

        public const int latBmp = 600;      // latura bitmapului
        public readonly List<Pixel> bmpPixels = new List<Pixel>() { };
        public const int dimPal = 1024;     // numarul de culori in paleta
        private Color[] paleta = new Color[dimPal];

        public Color PenColor { get; set; }     // culoarea scrisului
        public Color ScreenColor { get; set; }  // culoarea fondului

        private bool isWorking = false;  // pentru oprirea fortata  a desenarii


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

```
            //in vectorul paleta stocam dimPal culori de pe o curba inchisa din spatiul
RGB,
            //fixata in mod convenabil
            for (int k = 0; k < dimPal; k++)
            {
                double tcol = 56.123 + 2.0 * Math.PI * k / (double)dimPal;
                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);
            }

            //in lista bmpPixels stocam toti Pixel-ii corespunzatori PictureBox-ului
picBox

            for (int ii = 0; ii < latBmp; ii++)
            {
                for (int jj = 0; jj < latBmp; jj++)
                {
                    bmpPixels.Add(new Pixel(ii, jj));
                }
            }

            ScreenColor = Color.MidnightBlue;
            PenColor = Color.White;

            writeTitleBar("");
        }

        ..........................................................................

}
```

Remarcăm câmpul privat `myBmp`, un obiect de tip `Bitmap`, care conține o matrice de pixeli care pot fi accesți cu metoda `myBmp.SetPixel(int, int, Color)`. Dimensiunea matricei va fi setată la `latBmp x latBmp` pixeli, în acest exemplu `600 x 600` pixeli, dar programatorul poate modifica constanta `latBmp` la nevoie, forma adaptându-și dimensiunile, acestea fiind setate în constructor în funcție de `latBmp`, după cum se vede aici:

```
this.ClientSize = new System.Drawing.Size(latBmp + 25, latBmp + 39);
```

Bitmap-ul `myBmp` este folosit pentru păstrarea în memorie a pixelilor desenaţi în program, pentru a putea fi afişat el trebuie să fie mai întâi atribuit proprietăţii `Image` a picture-box-ului, atribuire făcută odată cu iniţializarea uneltelor grafice la încărcarea formei, în handler-ul de eveniment corespunzător:

```
        private void ComplexForm_Load(object sender, EventArgs e)
        {
            myBmp = new Bitmap(latBmp, latBmp, PixelFormat.Format24bppRgb);
            myBmp.SetResolution(300F, 300F);
            picBox.Image = myBmp;
            myDC = Graphics.FromImage(myBmp);
            myBrush = new SolidBrush(ScreenColor);
            myDC.FillRectangle(myBrush, 0, 0, latBmp, latBmp);
        }
```

În acest fel, orice re-pictare a suprafeței picture-box-ului provocată de `picBox.Refresh()` sau de `picBox.Invalidate()` va afișa pe ecran bitmapul `myBmp`.

5. Fișierul `Pixel.cs`. Pentru a facilita accesul la fiecare pixel al bitmapului, în spațiul de nume `ComplexExplorer` a fost definită structura

```
       public struct Pixel
       {
              public int I { get; set; }
              public int J { get; set; }
              public Pixel(int ii, int jj) : this()
              {
                     I = ii;
                     J = jj;
              }
       }
```

dotată, după cum se vede numai proprietățile publice auto-implementate `I` și `J`, de tip `int`, reprezentând coordonatele pixelului curent, mai precis indicii *i* și *j* ai elementului corespunzător lui în matricea bitmapului.

După cum am văzut mai sus, în `ComplexForm.cs`, clasa `ComplexForm` a fost dotată cu o listă de `Pixel`-i

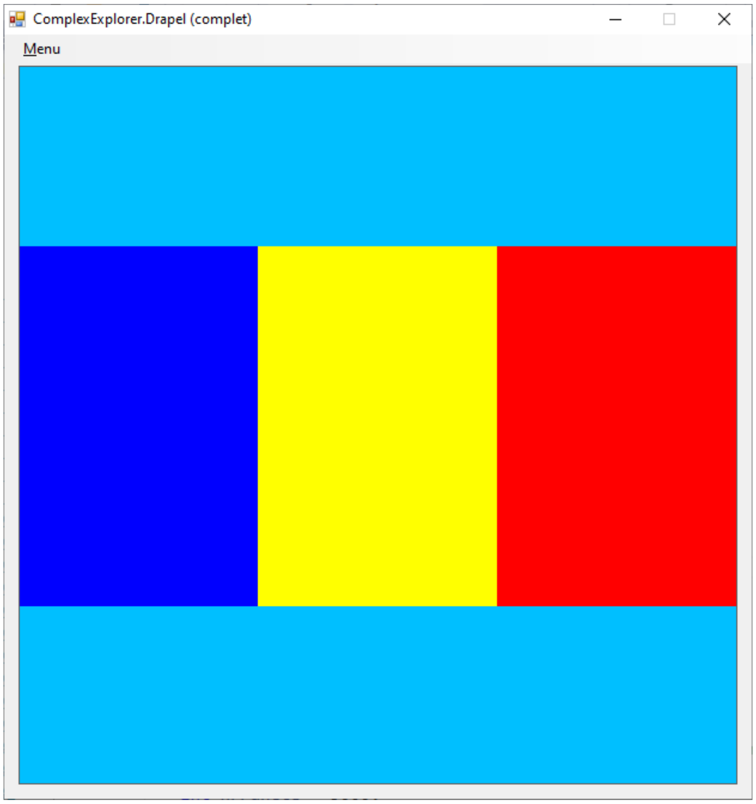
```
       public readonly List<Pixel> bmpPixels = new List<Pixel>() { };
```

care este încărcată în constructor cu toți pixelii reprezentabili în `myBmp`. În consecință, o modalitate rapidă de a colora toți pixelii din `myBmp`, este de a parcurge cu `foreach` lista `bmpPixels`

De exemplu, clasa următoare desenează tricolorul românesc:

```
    public class Drapel : ComplexForm
    {
        public override void makeImage()
        {
            foreach (Pixel p in bmpPixels)
            {
                Color col;
                if (p.J < jmax / 4 || p.J > 3 * jmax / 4) col = Color.DeepSkyBlue;
                else if (p.I < imax / 3) col = Color.Blue;
                else if (p.I < 2 * imax / 3) col = Color.Yellow;
                else col = Color.Red;
                setPixel(p, col);
            }
        }
```

```
}
```



Tot în fișierul Pixel.cs, pe lângă definiția structurii `Pixel`, a fost continuată definirea clasei `ComplexForm`, grupând aici uneltele ei de desenare:

```
public partial class ComplexForm : Form
{

    public const int imin = 0, imax = latBmp - 1, jmin = 0, jmax = latBmp - 1;
    private double xmin = 0.0, xmax = 1.0, ymin = 0.0, ymax = 1.0;
    private double dxdi = 1.0 / imax, dydj = 1.0 / jmax, didx = imax, djdy = jmax;

    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 void setPixel(int i, int j, Color c)
    {
```

```
            if (i < imin || imax < i || j < jmin || jmax < j) return;
            myBmp.SetPixel(i, jmax - j, c);
        }
        .....................................................
    }
```

În orice program de grafică bi-dimensională trebuie să utilizăm 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 planul *Oxy* 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 = latBmp - 1, jmin = 0, jmax = latBmp - 1;
```

în timp ce dreptunghiul din planul *Oxy* 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);
```

Aceste formule sunt folosite în metodele

```
public int getI(double x)
public int getJ(double y)
public Pixel getP(double x, double y)
public Pixel getP(Complex z)
public double getX(int i)
public double getY(int j)
public Complex getZ(int i, int j)
public Complex getZ(Pixel p)
```

care implementează transformările de coordonate sugerate de denumirea lor.

Pentru a preciza o culoare avem nevoie de un obiect din clasa `System.Drawing.Color`, de fapt o structură, inițializate de exemplu prin instrucțiuni de forma

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

Pentru a dispune de o paletă proprie de culori clasa `ComplexForm` a fost dotată cu un tablou de culori, `private Color[] paleta`, inițializat la încărcarea clasei în memorie,

```
public const int dimPal = 1024;     // numarul de culori in paleta
private Color[] paleta = new Color[dimPal];
```

și setat în constructorul formei. 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];
}
```

Metoda returnează culori indexate în mod continuu, din nuanța în nuanță.

În sfârșit, am ajuns și la desenarea pixel cu pixel. Metoda principală este

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

care a fost supraîncărcată în următoarele trei moduri:

```
public void setPixel(Pixel p, Color c)
public void setPixel(double x, double y, Color c)
public void setPixel(Complex z, Color c)
```

Sistemul de operare Windows afișează imaginile linie cu linie de sus în jos, exact cum apar textele în mod consolă, din acest motiv apare `jmax - j` în apelul `myBmp.SetPixel()`, noi formăm desenul răsturnat în bitmap iar Windows-ul îl mai răstoarnă o dată când îl afișează.

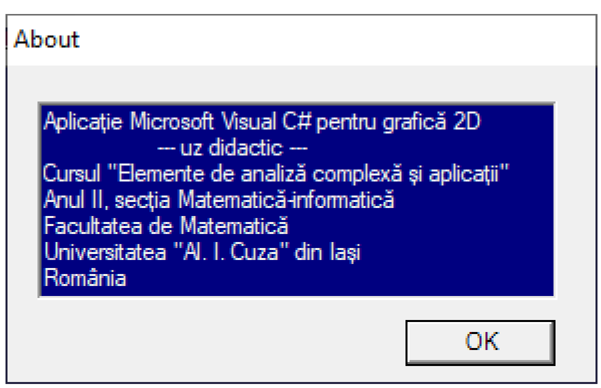
Segmentul dintre două puncte poate fi trasat, pixel cu pixel, prin metodele

```
public void setLine(int i0, int j0, int i1, int j1, Color c);
public void setLine(Pixel p0, Pixel p1, Color c)
public void setLine(double x0, double y0, double x1, double y1, Color c);
public void setLine(Complex z0, Complex z1, Color c);
```

Ultima metodă definită în acest fișier este utilzată pentru trasarea rapidă a axelor de coordonate

```
public void setAxis()
{
    setLine(xmin, 0.0, xmax, 0.0, PenColor);
    setLine(0.0, ymin, 0.0, ymax, PenColor);
}
```

6. Fișierul `AboutForm.cs`. La rulare, când este selectat itemul `About` din meniu, apare următoarea fereastră de dialog:



Această fereastră a fost implementată, din motive didactice, ca o formă secundară a cărui cod a fost scris în totalitate manual, fără ajutorul designerului:

```
using System;
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 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.Navy;
            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();
        }
    }
}
```

Forma principală `ComplexForm` a fost dotată cu un câmpul privat din clasa `AboutForm`

```
public partial class ComplexForm : Form
{
       private static AboutForm aboutForm = new AboutForm();
       .............
       private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
       {
              aboutForm.ShowDialog();
       }
       ................
}
```

prin intermediul căruia această clasă este arătată pe ecran la selectarea itemului corespunzător.

7. Tratarea evenimentelor. În codul de mai sus al clasei `AboutForm` observăm instrucțiunea

```
btnOK.Click += new EventHandler(btnOK_Click);
```

care adaugă la răspunsul prestabilit[2] al butonului la primirea mesajului "clic în dreptul tău"[3] încă un răspuns, și anume cel dat de metoda

```
private void btnOK_Click(object sender, EventArgs eArgs)
{
     this.Close();
}
```

care închide instanța curentă a clasei `AboutForm`, mai precis obiectul țintit de câmpul `aboutForm`.

Formele, butoanele și în general toate celelalte componentele care moștenesc clasa `System.Windows.Forms.Control` sunt dotate cu o serie de câmpuri speciale, numite evenimente (*events*), care sunt de fapt niște pointeri către metode.

Câmpul Click al clasei `Control` are declarația

```
public event EventHandler Click;
```

prin urmare `Click` este un eveniment de tip `EventHandler`, ceea ce însemnă ca lui îi pot fi atribuite sau adăugate numai metode cu semnătura

```
private void numeRaspuns(object expeditor, EventArgs date)
```

Apelul constructorului `new EventHandler(numeRaspuns)` returnează de fapt un pointer către metoda `numeRaspuns()` care poate fi atribuit evenimetului `Click`. În acest caz se spune că

2 de exemplu: să își re-picteze fereastra astfel încât să sugereze apăsarea și ridicarea

3 clic cu mausul sau cu tasta enter dacă butonul are focus.

`numeRaspuns()` este metoda delegată evenimentului `Click` sau, echivalent, că este handler-ul evenimentului `Click`.

Dacă studiem codul clasei `ComplexForm` vom afla din `ComplexForm.Designer.cs` că au fost actualizate următoarele evenimente corespunzătoare itemilor din menu:

```
this.startToolStripMenuItem.Click +=
   new System.EventHandler(this.startToolStripMenuItem_Click);
this.stopToolStripMenuItem.Click +=
   new System.EventHandler(this.stopToolStripMenuItem_Click);
this.saveToolStripMenuItem.Click +=
   new System.EventHandler(this.saveToolStripMenuItem_Click);
this.aboutToolStripMenuItem.Click +=
   new System.EventHandler(this.aboutToolStripMenuItem_Click);
this.exitToolStripMenuItem.Click +=
   new System.EventHandler(this.exitToolStripMenuItem_Click);
```

și următoarele trei evenimente ale formei

```
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.ComplexForm_KeyPress);
```

Codul metodelor delegate acestor evenimente se găsește în `ComplexForm.cs`, și poate fi studiat. De exemplu

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

}
```

arată că apăsarea barei de spațiu lansează desenarea dacă aceasta nu este deja în desfășurare și o oprește în caz contrar.

Anexe

1. Fişierul Program.cs

```
using System;
using System.Windows.Forms;

namespace ComplexExplorer
{

    static class Program
    {
        [STAThread]
        public static void Main()
        {
            Application.Run(new Sablon());
            //Application.Run(new RandomList());
            //Application.Run(new SistemDinamic());
            //Application.Run(new CurbeLissajous());
            //Application.Run(new Pixeli());
        }
    }
}
```

2. Fişierul MyToolbox.cs

```
using System;
using System.Drawing;
using System.Collections.Generic;


namespace ComplexExplorer{

    public class Sablon : ComplexForm
    {
        Complex[] FiguraInitiala(double r)  // un cerc care trece prin origine
        {
            int nrPuncte = 5000;
            Complex[] rez = new Complex[nrPuncte + 1];
            double delta = 2 * Math.PI / nrPuncte;
            for (int k = 0; k <= nrPuncte; k++)
            {
                rez[k] = r + Complex.setRoTheta(r, k * delta);
            }
            return rez;
        }
```

```
    public override void makeImage()
    {
        double lat = 2;
        setXminXmaxYminYmax(-lat, lat, -lat, lat);
        setAxis();
        Complex[] fig = FiguraInitiala(lat/3);
        Complex rotor = Complex.setRoTheta(1, Math.PI / 24);
        for (int t = 0; t < 10000; t++)
        {
            //initScreen();
            Color col = getColor(2*t);
            for (int k = 0; k < fig.Length; k++)
            {
                setPixel(fig[k], col);
                fig[k] *= rotor;
            }
            if (!resetScreen()) return;

        }
        setAxis();
        resetScreen();
    }
}

/************************************************************/


public class RandomList : ComplexForm
{
    Random zar = new Random();
    double MyNext()
    {
        return (zar.NextDouble() - 0.5) / 5; //[-0.1,+0.1]
    }

    Complex T(Complex z)
    {
        return z + Complex.setRoTheta(1 + 6 * MyNext(), z.Theta + MyNext());
    }
    List<Complex> FiguraInitiala()     // un poligon regulat centrat in origine
    {
        var rez = new List<Complex>();
        int nrPuncte = 200;
        double ro = 1, delta = 2 * Math.PI / nrPuncte;
        for (int k = 0; k <= nrPuncte; k++)
        {
            rez.Add(Complex.setRoTheta(ro, k * delta));
        }
        return rez;
    }
```

```
    void Transforma(ref List<Complex> fig)
    {
        var rez = new List<Complex>();
        foreach (Complex z in fig)
        {
            rez.Add(T(z));
            if (zar.Next(100) < 16) rez.Add(T(z));
        }
        fig = rez;
    }

    void Traseaza(List<Complex> li, Color col)
    {

        for (int k = 1; k < li.Count; k++)
        {
            setLine(li[k - 1], li[k], col);
        }

    }

    public override void makeImage()
    {
        double r = 20;
        setXminXmaxYminYmax(-r, r, -r, r);
        List<Complex> fig = FiguraInitiala();
        int k0 = zar.Next(100);
        initScreen();
        Traseaza(fig, getColor(k0));
        for (int k = 0; k < 15; k++)
        {
            //initScreen();
            Transforma(ref fig);
            Traseaza(fig, getColor(k0 + 20 * k));
            setTextAt(new Complex(-r, r - k), fig.Count.ToString());
            if (!resetScreen()) return;
        }
    }
}



/*****************************************************/
public class SistemDinamic : ComplexForm
{

    public struct PunctColorat
    {
```

```
            public Complex Z { get; set; }
            public Color Col { get; set; }
            public PunctColorat(Complex z, Color col)
                : this()
            {
                Z = z;
                Col = col;
            }

        }
        public void setPixel(PunctColorat p)
        {
            setPixel(p.Z, p.Col);
        }
        const double a = -10, b = 1;
        const double c = -1, d = 10;
        //const  double a = -1, b = 5;
        //const  double c = -5, d = -1;

        const double h = 0.005;
        const double raza = 2;
        Random zar = new Random();

        PunctColorat PunctAleator()
        {
            int iz = zar.Next(latBmp);
            int jz = zar.Next(latBmp);
            return new PunctColorat(getZ(iz, jz), getColor((iz + jz) / 2 +
zar.Next(100)));

        }

        void UnPas(PunctColorat[] puncte)
        {
            for (int k = 0; k < puncte.Length; k++)
            {
                Complex zVechi = puncte[k].Z;
                double xv = zVechi.Re;
                double yv = zVechi.Im;

                double xn = xv + h * Math.Sin(a * xv + b * yv);
                double yn = yv + h * Math.Sin(c * xv + d * yv);

                Complex zNou = new Complex(xn, yn);

                if (zNou.Ro < raza && (zNou - zVechi).Ro2 > 1.0e-8) puncte[k].Z = zNou;
                else puncte[k] = PunctAleator();
                setPixel(puncte[k]);
            }
```

```
    }
    public override void makeImage()
    {
        setXminXmaxYminYmax(-raza / 2, raza / 2, -raza / 2, raza / 2);
        int nrPuncte = 10000;
        var tab = new PunctColorat[nrPuncte];
        for (int k = 0; k < nrPuncte; k++)
        {
            tab[k] = PunctAleator();
        }
        for (; resetScreen();)
        {
            initScreen();
            UnPas(tab);
        }
    }
}


/****************************************************/
public class CurbeLissajous : ComplexForm
{
    public override void makeImage()
    {
        setXminXmaxYminYmax(-1.3, 1.3, -1.3, 1.3);
        const double omega1 = 0.001;
        const double omega2 = 1.5007 * omega1;
        const double fi = 0.1;
        for (int t = 0; ; t++)
        {
            double x = Math.Sin(omega1 * t);
            double y = Math.Sin(omega2 * t + fi);
            setPixel(x, y, getColor(t / 5000));
            if (t % 10000 == 0 && !resetScreen()) return;
        }
    }
}

/****************************************************/
public class Pixeli : ComplexForm
{
    Complex i = new Complex(0, 1);

    public override void makeImage()
    {
        Complex q = 4 + 3 * i;
        double r = 5;
        setXminXmaxYminYmax(q, r + 4);
        PenColor = Color.White;
        setAxis();
```

```
            Color col;
            foreach (Pixel p in bmpPixels)
            {
                Complex z = getZ(p);
                bool esteSus = (z.Im > 0);
                bool esteLaDreapta = (z.Re > 0);
                bool esteInDisc = ((z - q).Ro < r);

                if (esteSus)
                    col = (esteLaDreapta ? Color.SkyBlue : Color.DeepSkyBlue);
                else
                    col = (esteLaDreapta ? Color.DarkTurquoise : Color.LightSeaGreen);
                if (esteInDisc)
                    col = (esteSus ? getColor(100 + p.J / 2) : getColor(600 + p.J));
                setPixel(p, col);
                if (p.J == 0 && !resetScreen()) return;
            }

            Pixel est = getP(q + r), vest = getP(q - r);
            setLine(est, vest, Color.Magenta);
            setLine(q + r * i, q - r * i, Color.Yellow);
            PenColor = Color.MidnightBlue;
            setAxis();
            resetScreen();
        }
    }

}
```

3. 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;
otherwise, 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.startToolStripMenuItem_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.stopToolStripMenuItem_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.saveToolStripMenuItem_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.aboutToolStripMenuItem_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.exitToolStripMenuItem_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.ComplexForm_KeyPress);
            ((System.ComponentModel.ISupportInitialize)(this.picBox)).EndInit();
            this.mainMenu.ResumeLayout(false);
            this.mainMenu.PerformLayout();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private 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;
    }
}
```

4. Fişierul ComplexForm.cs

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

namespace ComplexExplorer
{
    public partial class ComplexForm : Form
    {
        private static AboutForm aboutForm = new AboutForm();
        private Graphics myDC;  // my device context
        private Bitmap myBmp;   // my bitmap
        private SolidBrush myBrush;

        public const int latBmp = 600;      // latura bitmapului
        public readonly List<Pixel> bmpPixels = new List<Pixel>() { };
        public const int dimPal = 1024;     // numarul de culori in paleta
        private Color[] paleta = new Color[dimPal];

        public Color PenColor { get; set; }     // culoarea scrisului
        public Color ScreenColor { get; set; }  // culoarea fondului

        private bool isWorking = false;  // pentru oprirea fortata  a desenarii


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

            //in vectorul paleta stocam dimPal culori de pe o curba inchisa din spatiul
RGB,
            //fixata in mod convenabil
            for (int k = 0; k < dimPal; k++)
```

```
            {
                double tcol = 56.123 + 2.0 * Math.PI * k / (double)dimPal;
                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);
            }

            //in lista bmpPixels stocam toti Pixel-ii corespunzatori PictureBox-ului
picBox

            for (int ii = 0; ii < latBmp; ii++)
            {
                for (int jj = 0; jj < latBmp; jj++)
                {
                    bmpPixels.Add(new Pixel(ii, jj));
                }
            }

            ScreenColor = Color.MidnightBlue;
            PenColor = Color.White;

            writeTitleBar("");
        }

        private void ComplexForm_Load(object sender, EventArgs e)
        {
            myBmp = new Bitmap(latBmp, latBmp, PixelFormat.Format24bppRgb);
            myBmp.SetResolution(300F, 300F);
            picBox.Image = myBmp;
            myDC = Graphics.FromImage(myBmp);
            myBrush = new SolidBrush(ScreenColor);
            myDC.FillRectangle(myBrush, 0, 0, latBmp, latBmp);
        }

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

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

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

```
        for (int k = 0; k < mesaj.Length; k++)
            myDC.DrawString(mesaj[k], font, myBrush, 0.0F, k * 15.0F);
    }
}
public void setTextAt(Complex z, string msg)
{
    myBrush.Color = PenColor;
    using (Font font = new Font("Arial", 3.2f))
    {
        myDC.DrawString(msg, font, myBrush, getI(z.Re), jmax - getJ(z.Im));
    }
}
public void initScreen()
{
    myBrush.Color = ScreenColor;
    myDC.FillRectangle(myBrush, 0, 0, latBmp, latBmp);
    picBox.Invalidate();
}

public bool resetScreen()
{
    picBox.Refresh();           //pentru re-pictarea ferestrei
    Application.DoEvents();     //pentru procesarea mesajelor
    return isWorking;           //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();
    isWorking = true;
    writeTitleBar("in lucru");
    makeImage();
    if (isWorking) writeTitleBar("complet");
    else writeTitleBar("oprit");
    isWorking = false;
    this.startToolStripMenuItem.Enabled = true;
    this.stopToolStripMenuItem.Enabled = false;
    this.saveToolStripMenuItem.Enabled = true;
}
```

```
        private void stopToolStripMenuItem_Click(object sender, EventArgs e)
        {
            isWorking = false;
            this.startToolStripMenuItem.Enabled = true;
            this.stopToolStripMenuItem.Enabled = false;
            this.saveToolStripMenuItem.Enabled = true;
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            isWorking = false;
            this.Close();
        }

        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (myBmp == null) return;

            mySaveFileDialog.Filter = "Bitmap Image (.bmp)|*.bmp|JPEG Image (.jpg)|*.jpg|
Png Image (.png)|*.png";


            if (mySaveFileDialog.ShowDialog() != DialogResult.OK) return;
            string numeFisier = mySaveFileDialog.FileName;
            if (numeFisier.Length > 0)
            {
                switch (mySaveFileDialog.FilterIndex)
                {
                    case 1:
                        numeFisier = System.IO.Path.ChangeExtension(numeFisier, ".bmp");
                        myBmp.Save(numeFisier, ImageFormat.Bmp);
                        break;
                    case 2:
                        numeFisier = System.IO.Path.ChangeExtension(numeFisier, ".jpg");
                        myBmp.Save(numeFisier, ImageFormat.Jpeg);
                        break;
                    case 3:
                        numeFisier = System.IO.Path.ChangeExtension(numeFisier, ".png");
                        myBmp.Save(numeFisier, ImageFormat.Png);
                        break;
                }
            }
        }
        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 (!isWorking) 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));
        }

    }

}
```

5. Fişierul Pixels.cs

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


namespace ComplexExplorer
{
    public struct Pixel
    {
        public int I { get; set; }
        public int J { get; set; }
        public Pixel(int ii, int jj) : this()
        {
            I = ii;
            J = jj;
        }
    }

    public partial class ComplexForm : Form
    {

        public const int imin = 0, imax = latBmp - 1, jmin = 0, jmax = latBmp - 1;
        private double xmin = 0.0, xmax = 1.0, ymin = 0.0, ymax = 1.0;
        private double dxdi = 1.0 / imax, dydj = 1.0 / jmax, didx = imax, djdy = jmax;


        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 void setXminXmaxYminYmax(Complex q, double r)
        {
            setXminXmaxYminYmax(q.Re - r, q.Re + r, q.Im - r, q.Im + r);
        }

        public int getI(double x)
        {
            return (int)Math.Round(imin + (x - xmin) * didx);
        }

        public int getJ(double y)
        {
            return (int)Math.Round(jmin + (y - ymin) * djdy);
        }

        public Pixel getP(double x, double y)
        {
            return new Pixel((int)Math.Round(imin + (x - xmin) * didx),
(int)Math.Round(jmin + (y - ymin) * djdy));
        }
        public Pixel getP(Complex z)
        {
            return new Pixel((int)Math.Round(imin + (z.Re - xmin) * didx),
(int)Math.Round(jmin + (z.Im - 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 Complex getZ(Pixel p)
        {
            return new Complex(xmin + (p.I - imin) * dxdi, ymin + (p.J - jmin) * dydj);
```

```
        }

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

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


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

        public void setPixel(Complex z, Color c)
        {
            setPixel((int)Math.Round(imin + (z.Re - xmin) * didx), (int)Math.Round(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(Pixel p0, Pixel p1, Color c)
        {
            setLine(p0.I, p0.J, p1.I, p1.J, c);
        }

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

        public void setLine(Complex z0, Complex z1, Color c)
        {
            setLine((int)Math.Round(imin + (z0.Re - xmin) * didx), (int)Math.Round(jmin +
(z0.Im - ymin) * djdy),
                (int)Math.Round(imin + (z1.Re - xmin) * didx), (int)Math.Round(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);
        }
    }
}
```

6. Fişierul AboutForm.cs

```
using System;
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 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.Navy;
            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();
        }
    }
}
```

7. 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);
    }
    set
    {
        x = value * Math.Cos(Theta);
        y = value * Math.Sin(Theta);
    }
}
public double Ro2
{
    get
    {
        return x * x + y * y;
    }
}
public double Theta
{
    get
    {
        return Math.Atan2(y, x);
    }
    set
    {
        x = Ro * Math.Cos(value);
        y = Ro * Math.Sin(value);
    }
}
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);
    }
    set
    {
        x = value.x;
        y = -value.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;
            if (r < 1.0e-12) return Complex.setRoTheta(Double.MaxValue, zst.Theta -
zdr.Theta);
            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();
        }
    }
}
```