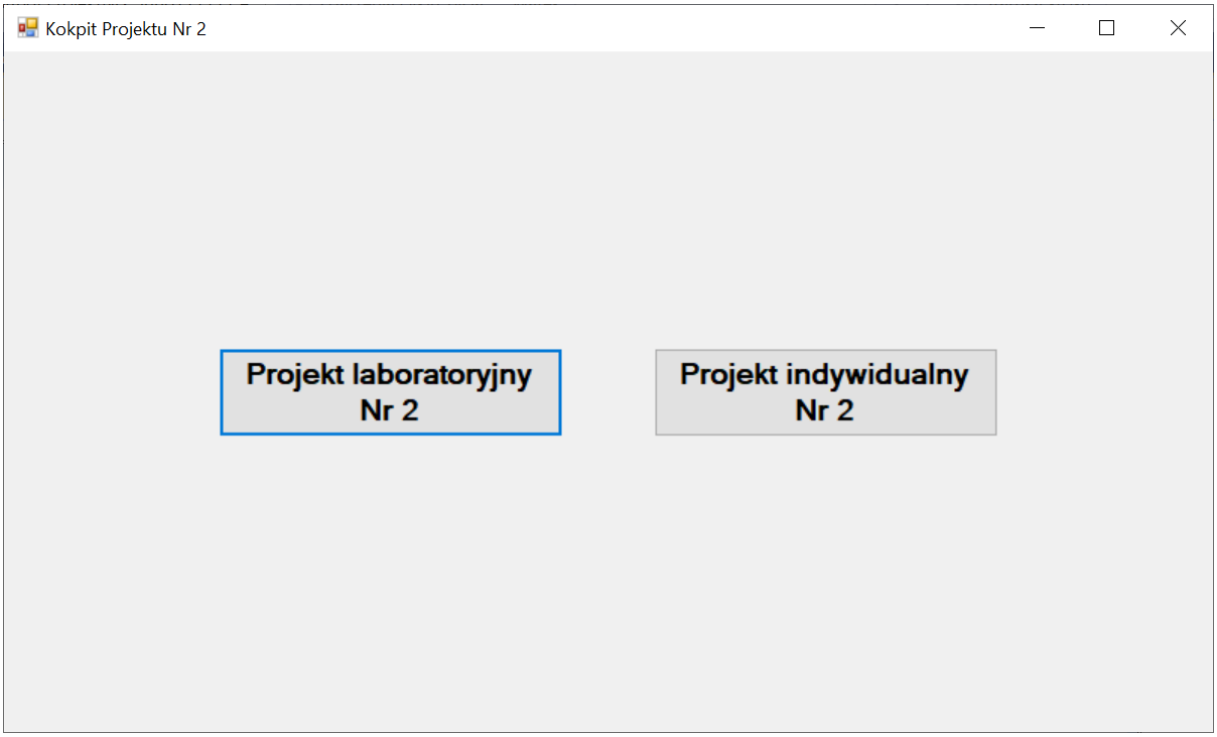
Opracował: Leszek Jung, **AFiBV**: **Wydział Informatyki, Grafiki i Architektury**,
kierunek studiów: INFORMATYKA, (http://vistula.edu.pl)
Przedmiot: Programowanie obiektowe

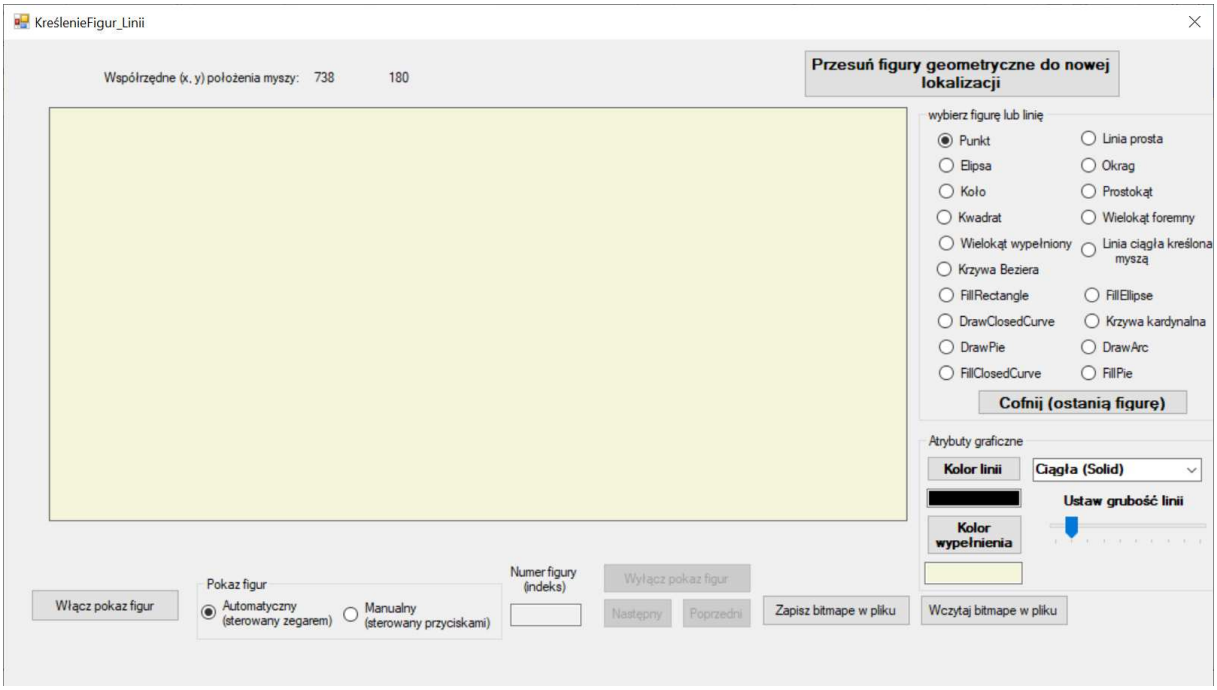
Obsługa Myszy

Po zaprojektowaniu formularza głównego (o nazwie: KokpitProjektuNr2):



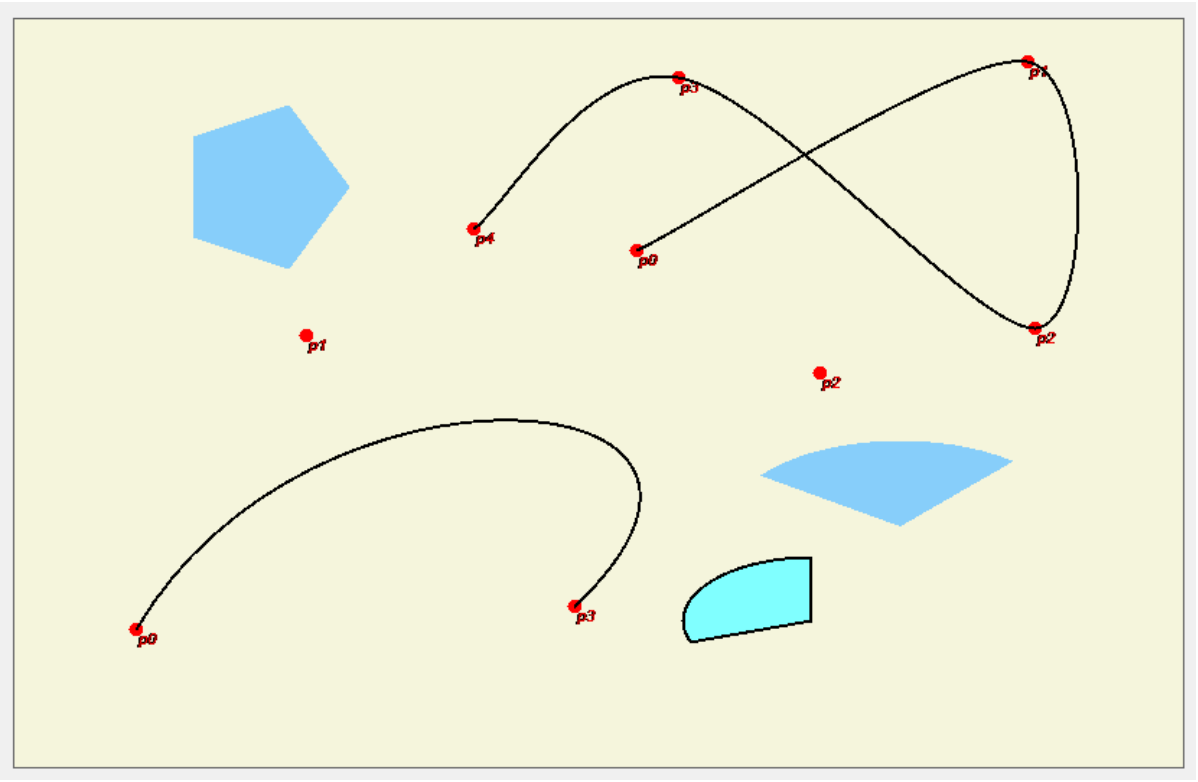
Przystępujemy do zaprojektowania formularza : Projekt indywidulany Nr2 (do którego przechodzimy po wybraniu przycisku poleceń: Projekt indywidulany Nr2), którego interfejs graficzny mógłby przyjąć postać:

Opracował: Leszek Jung, **AFiBV**: **Wydział Informatyki, Grafiki i Architektury**,
kierunek studiów: INFORMATYKA, (http://vistula.edu.pl)
Przedmiot: Programowanie obiektowe



Narzędzia (kontrolki i Mysz) powinny umożliwić kreślenie figur i linii geometrycznych przy użyciu Myszy:

Opracował: Leszek Jung, **AFiBV**: **Wydział Informatyki, Grafiki i Architektury**,
kierunek studiów: INFORMATYKA, (http://vistula.edu.pl)



Dla kreślenia figur i linii geometrycznych na Rysownicy przy użyciu Myszy musimy poznać właściwości kontrolki `PictureBox` (reprezentującej Rysownicę) oraz technikę obsługi zdarzeń od Myszy, które sa rozpoznawane przez kontrolkę `PictureBox`.

Rysownica

Rysownicę (czyli powierzchnię graficzną, na której będzie kreślili figury i linii geometryczne) tworzymy „na" kontrolce PictureBox, która:

- umożliwia wyświetlanie obrazów graficznych w formacie (wybór formatu zależy od specyfiki pliku graficznego): BMP (bitmapy), ICO (ikony), WMF (metapliki),

  JPG/JPEG (grafika rastrowa z kompresją stratną, co oznacza, że obraz po zmniejszeniu traci na jakości, szeroki zakres barw i płynne przejścia tonalne, zalecany do obrazów naturalnych, czyli zdjęć),

  GIF (grafika rastrowa i kompresja bezstratna, obsługuje pełną przezroczystość obrazu i umożliwia tworzenie animacji poklatkowej i stąd duża popularność tych plików),

  PNG (grafika rastrowa z kompresją stratną, zapewnia swobodę w modyfikacji rozmiaru obrazu bez utraty jakości, obsługuje przezroczystość stopniowaną, umożliwia zapisywanie obrazów 24 –bitowych, nie stosuje się w fotografii, gdyż pliki są dużo większe niż w przypadku formatu JPG),

- ma właściwość Image, której można przypisać mapę bitową:

```
pbRysownica.Image = new Bitmap(pbRysownica.Width, pbRysownica.Height);
```

po czym możemy już utworzyć egzemplarz powierzchni graficznej:

```
Rysownica = Graphics.FromImage(pbRysownica.Image);
```

- rozpoznaje podstawowe zdarzenia generowane przez Mysz:

MouseClick
MouseDoubleClick
MouseDown
MouseEnter
MouseHover
MouseLeave
MouseMove
MouseUp
Move

Klasa BitMap

Klasa Bitmap, dziedzicząca właściwości po klasie abstrakcyjnej Image, jest często stosowana jako tło dla okna lub strony internetowej, jak też dla tworzenia tzw. małej grafiki na paskach narzędzi:

Klasa Bitmap:

- jest zdefiniowana w przestrzeni nazw System.Drawing,
- posiada dwie właściwości, które są szczególnie przydatne przy pracy z plikami graficznymi:
  - GetPixel: pobiera kolor wybranego piksela.
  - SetPixel –ustawia kolor wybranego piksela.

Egzemplarz powierzchni graficznej tworzymy na BitMap'ie przy użyciu metody FromImage, którą udostępnia klasa Graphics.

Kontrolka PictureBox, umieszczona na formularzu, będzie miała domyślnie ustawione przezroczyste tło, co oczywiście można zmienić w jej właściwościach (ustawiając BackColor na wymagany kolor).

Implementacji fragmentów programu kreślenia Myszą figur i linii geometrycznych

W widoku kodu formularza LaboratoriumNr2 (opisującej formularz: Projekt indywidualny Nr 2 (prezentacja figur i linii geometrycznych kreślonych Myszą) powinniśmy (dla ułatwienia zapisu dostępu do jej składników) udostępnić przestrzeń nazw Drawing2D (dla potrzeb grafiki 2D):

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
// dodanie przestrzeni nazw dla potrzeb grafiki 2D
using System.Drawing.Drawing2D;
```

W klasie LaboratoriumNr2 (opisującej formularz: Projekt indywidualny Nr 2 (prezentacja figur i linii geometrycznych kreślonych Myszą) deklarujemy stałe i zmienne referencyjne narzędzi graficznych:

```
public partial class LaboratoriumNr2 : Form
{
    // deklaracje pomocnicze
    const ushort Margines = 10; // odstęp od krawędzi Rysownicy
    const ushort MarginesFormularza = 20;
    // deklaracja powierzchni graficznej
    Graphics Rysownica;
    // deklaracja pióra
    Pen Pióro;
    // deklaracja Pędzla
    SolidBrush Pędzel;
```

W konstruktorze formularza LaboratoriumNr2 ustalamy lokalizację i rozmiar samego formularza:

```
public LaboratoriumNr2()
{ InitializeComponent();
  // lokalizacja i zwymiarowanie formularza
  this.Location =
    new Point(Screen.PrimaryScreen.Bounds.X + MarginesFormularza,
         Screen.PrimaryScreen.Bounds.Y + 2 * MarginesFormularza);
  this.Width = (int)(Screen.PrimaryScreen.Bounds.Width * 0.9F);
  this.Height = (int)(Screen.PrimaryScreen.Bounds.Height * 0.85F);
  this.StartPosition = FormStartPosition.Manual;
  // "usuwamy" przyciski Minimalizacji i Maksymalizacji okan formularza
  this.MaximizeBox = false;
  this.MinimizeBox = false;
  this.AutoScroll = true;
```

W konstruktorze formularza LaboratoriumNr2 ustalamy również lokalizację i rozmiar PictureBox (o nazwie pbRysownica) oraz tworzymy egzemplarz powierzchni graficznej oraz tworzymy egzemplarze narzędzi graficznych (Rysownica, Pióro, Pędzel, . . .):

```
// lokalizacja i zwymiarowanie kontrolki PictureBox
pbRysownica.Location = new Point(Left + 2 * MarginesFormularza,
                                 Top + 2 *  MarginesFormularza);
pbRysownica.Width = (int)(this.Width * 0.7F);
pbRysownica.Height = (int)(this.Height * 0.8F);
pbRysownica.BackColor = Color.Beige;
pbRysownica.BorderStyle = BorderStyle.FixedSingle;
// utworzenie mapy itowej i podpięcie jej do kontrolki PictureBox
pbRysownica.Image = new Bitmap(pbRysownica.Width, pbRysownica.Height);
// utworzenie egzemplarza powierzchni graficznej na BitMapie
Rysownica = Graphics.FromImage(pbRysownica.Image);
```

W konstruktorze formularza LaboratoriumNr2 tworzymy również egzemplarze narzędzi graficznych (Pióro, Pędzel):

```
// utworzenie egzemplarza pióra głównego
Pióro = new Pen(Color.Black, 1F);
// formatowanie Pióra
Pióro.DashStyle = DashStyle.Solid; // linia ciągła
Pióro.StartCap = LineCap.Round; // zaokrąglenie
Pióro.EndCap = LineCap.Round; // zaokrąglenie
// utworzenie egzemplarza Pędzla
Pędzel = new SolidBrush(Color.Blue); // tzw. pędzel ciągły
```

Kreślenie Myszą figur i linii geometrycznych

Podstawową techniką programowania w projektowaniu aplikacji formularzowych jest *programowanie sterowane zdarzeniami* (ang. *event-driven programming*), w którym wywołanie metod obsługi zdarzenia następuje automatycznie po zaistnieniu danego zdarzenia.

Jeśli mamy umożliwić kreślenia figur i linii Myszą, to musimy oprogramować zdarzenia:

- naciśnięcia lewego przycisku myszy (MouseDown) dla „odnotowania" początku kreślonej linii,
- przesuwania myszy (MouseMove), przy wciśniętym lewym przycisku myszy. dla kreślenia linii „podążającej za kursorem" myszy,
- zwolnienia lewego przycisku myszy(MouseUp).

Obsługa zdarzenia MouseDown

Przechodzimy więc do widoku formularza, zaznaczamy kontrolkę pbRysownica, a w oknie właściwości wybieramy (kliknięciem zakładkę Events:

- przewijamy listę zdarzeń (prawdopodobnych) dla pbRysownica i odszukujemy zdarzenie MouseDowna,
- klikamy dwukrotnie nazwę zdarzenia (MouseDown) ,
- w oknie edytora Visual Studio zostanie otwarty szablon metody obsługi zdarzenia MouseDown:

```
private void pbRysownica_MouseDown(object sender, MouseEventArgs e)
{

}
```

Kliknięcie lewym przyciskiem myszy traktujemy jako określenie lokalizacji figury geometrycznej lub linii, które będą wykreślane. Obsługa zdarzenia MouseDown powinna zapewnić zapamiętanie współrzędnych punktu kliknięcia (lewym przyciskiem myszy), które powinny być dostępne dla metod obsługi innych zdarzeń Myszy.
Oznacza to, że powinniśmy zadeklarować zmienną Punkt (dla przykładu) w części deklaracyjnej formularza LaboratoriumNr2 (opisującej formularz: Projekt indywidualny Nr 2 (prezentacja figur i linii geometrycznych kreślonych Myszą):


```
public partial class LaboratoriumNr2 : Form
{
    // deklaracje pomocnicze
    const ushort Margines = 10; // odstęp od krawędzi Rysownicy
    const ushort MarginesFormularza = 20;
    // deklaracja powierzchni graficznej
    Graphics Rysownica;
    // deklaracja pióra
    Pen Pióro;
    // deklaracja Pędzla
    SolidBrush Pędzel;
    /* deklaracja Punktu, któremu przypiszemy współrzędne (x, y) przy
     wciśnięciu lewego przycisku myszy */
    Point Punkt = Point.Empty;
```

W treści metody obsługi zdarzenia MouseDown zapamiętujemy współrzędne punktu kliknięcia lewym przyciskiem myszy.

```
private void pbRysownica_MouseDown(object sender,
                                   MouseEventArgs e)
{ // wypisanie współrzędnych X i Y położenia myszy
  lblX.Text = e.Location.X.ToString();
  lblY.Text = e.Location.Y.ToString();
  // obsługa zdarzeń od naciśniętego lewego przycisku
 if (e.Button == MouseButtons.Left)
   {  /* zapamiętanie (save) współrzędnych punktu, który
         został  kliknięty lewym przyciskiem myszy  */
      Punkt = e.Location;
   }
}
```

Dane (jaki przycisk i współrzędne położenia) o zaistniałym zdarzeniu są nam przekazywane przez parametr `e` w liście parametrów metody obsługi zdarzenia `MouseDown`:

```
private void pbRysownica_MouseDown(object sender, MouseEventArgs e)
{
    // wypisanie współrzędnych X i Y położenia myszy
    lblX.Text = e.
```


Obsługa zdarzenia MouseUp

Obsługa zdarzenia `MouseUp` (zwolnienia lewego przycisku myszy) powinna skutkować wykreśleniem wybranej figury (lub linii) geometrycznej:

```
private void pbRysownica_MouseUp(object sender,
                                 MouseEventArgs e)
{// wizualizacja (wypisanie) aktualnego położenia myszy
   lblX.Text = e.Location.X.ToString();
   lblY.Text = e.Location.Y.ToString();
 /* deklaracje zmiennych pomocniczych i wyznaczenie
     parametrów  opisujących prostokąt, w którym będzie
    wykreślana figura geometryczna */
```


```
int lewyGórnyNarożnikX =
        (Punkt.X > e.Location.X) ? e.Location.X : Punkt.X;
int lewyGórnyNarożnikY =
         (Punkt.Y > e.Location.Y) ? e.Location.Y : Punkt.Y;
int Szerokość = Math.Abs(Punkt.X - e.Location.X);
int Wysokość = Math.Abs(Punkt.Y - e.Location.Y);
 /* sprawdzenie, czy obsługiwane zdarzenie  MouseUp zostało
    spowodowane zwolnieniem lewego przycisku myszy */
if (e.Button == MouseButtons.Left)
 {
 // rozpoznanie figury lub linii geometrycznej i jej
                                                  wykreślenie
   .    .    .
  }
```

Po rozpoznaniu wybranej (aktualnie kreślonej) figury (lub linii) geometrycznej i jej wykreśleniu musimy odświeżyć powierzchnię graficzną kontrolki PictureBox o nazwie pbRysownica:

```
pbRysownica.Refresh();
```

Co zapisujemy po rozpoznaniu i wykreśleniu wybranej figury (lub linii) geometrycznej:

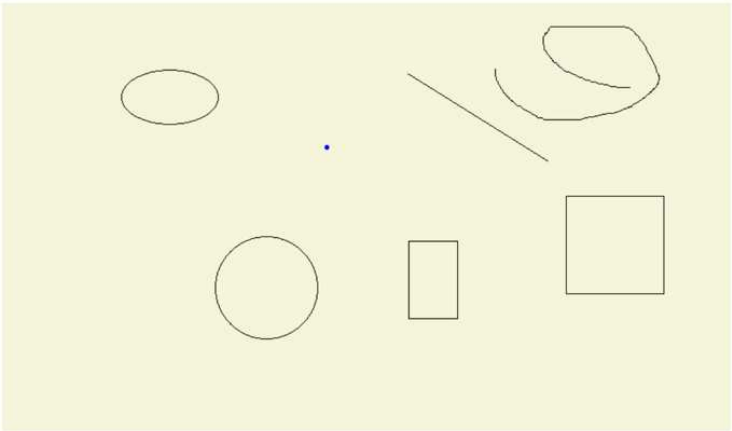
```
if (e.Button == MouseButtons.Left)
 {
 // rozpoznanie figury lub linii geometrycznej i jej
                                                  wykreślenie
   .    .    .
   // odświeżenie powierzchni graficznej
   pbRysownica.Refresh();
  }
```

A teraz rozpoznajemy figury (lub linie) geometryczne:

```
if (rdbPunkt.Checked)
  { // wykreślenie punktu jako wypełnionego okręgu
      Rysownica.FillEllipse(Pędzel,
                        Punkt.X - Pióro.Width / 2,
                        Punkt.Y - Pióro.Width / 2,
                         5* Pióro.Width, 5* Pióro.Width);
  }
if (rdbLinia.Checked)
  { // wykreślenie linii
    Rysownica.DrawLine(Pióro, Punkt.X, Punkt.Y,
                               e.Location.X, e.Location.Y);
  }

if (rdbElipsa.Checked) // wykreślenie elipsy
     Rysownica.DrawEllipse(Pióro, lewyGórnyNarożnikX,
          lewyGórnyNarożnikY, Szerokość, Wysokość);

 if (rdbOkrąg.Checked) // wykreślenie kręgu
    Rysownica.DrawEllipse(Pióro, lewyGórnyNarożnikX,
       lewyGórnyNarożnikY, Szerokość, Szerokość);

if (rdbProstokąt.Checked) // wykreślenie prostokątu
    Rysownica.DrawRectangle(Pióro,
        new Rectangle(lewyGórnyNarożnikX,
                      lewyGórnyNarożnikY,
                     Szerokość, Wysokość));
```


```
if (rdbKwadrat.Checked)
    Rysownica.DrawRectangle(Pióro,
   new Rectangle(lewyGórnyNarożnikX, lewyGórnyNarożnikY,
                        Szerokość, Szerokość));

if (rdbLiniaKreślonaMyszą.Checked)
 { // kreślenie linii ciągłej
   Rysownica.DrawLine(Pióro, Punkt.X, Punkt.Y,
                           e.Location.X, e.Location.Y);
   // uaktualnienie początku dla następnego odcinka linii
   Punkt.X = e.Location.X;
   Punkt.Y = e.Location.Y;
 }
 . . .
```

Możemy już uruchomić nasz program (wcześniej klikamy `Zapisz wszystko`) i wykreślać wybrane figury (lub linie) geometryczne:



Wizualny efekt rozciągania kreślonej figury geometrycznej

Obecna wersja projektu naszego programu ma zaimplementowaną obsługę dwóch zdarzeń myszy (MouseDown i MouseUp) dla kontrolki PictureBox, co „zubaża wizualnie" sam proces kreślenia figur geometrycznych.

Efekt wizualnego rozciągania kreślonej figury geometrycznej wymaga zaprojektowania metody obsługi zdarzenia MouseMove, w której, wybrana figura (lub linia) będzie kreślona na przezroczystej powierzchni graficznej kontrolki PictureBox.

Celem uzyskania wizualnego efektu „rozciąganego" kreślenia figury geometrycznej, w klasie LaboratoriumNr2 (opisującej formularz: Projekt indywidualny Nr 2 (prezentacja figur i linii geometrycznych kreślonych Myszą): :

1. deklarujemy zmienne referencyjne tymczasowej powierzchnii graficznej i tymczasowego pióra:

```
public partial class LaboratoriumNr2 : Form
    {   . . .
     /* utworzenie  Rysownicy tymczasowej na powierzchni
         kontrolki PictureBox  */
        Graphics RysownicaTymczasowa;
     /* deklaracja Pióra tymczasowego dla kreślenia na
            powierzchni tymczasowej */
        Pen PióroTymczasowe;
```

2. w konstruktorze klasy LaboratoriumNr2, tworzymy egzemplarz tymczasowej powierzchni graficznej i egzemplarz tymczasowego pióra :

```
public LaboratoriumNr2()
 { InitializeComponent();
    .    .    .
   // utworzenie egzemplarza Pędzla
   Pędzel = new SolidBrush(Color.Blue); // tzw. pędzel
                                                ciągły
/* utworzenie egzemplarza tymczasowej powierzchni
  graficznej na powierzchni przezroczystej kontrolki
  PictureBox */
  RysownicaTymczasowa = pbRysownica.CreateGraphics();
  /* utworzenie egzemplarza pióra tymczasowego (niebieskiego) dla
  wizualizacji "rozciągania" kreślonej figury geometrycznej   */
  PióroTymczasowe = new Pen(Color.Blue, 1);
```

Metoda CreateGraphics() klasy Control, jest dziedziczona przez wszystkie klasy pochodne klasy bazowej Control, a to oznacza, że egzemplarz powierzchni graficznej możemy utworzyć na powierzchni dowolnej kontrolki, czyli również, na powierzchni kontrolki PictureBox.

3. Projektujemy metodę obsługi zdarzenia MouseMove (dla kontrolki PictureBox), w której figury i linie geometryczne będą kreślone na tymczasowej powierzchni graficznej RysownicaTymczasowa przy użyciu PióroTymczasowe.

Obsługa zdarzenia MouseMove

```
private void pbRysownica_MouseMove(object sender,  MouseEventArgs e)
 { // wizualizacja (wypisanie) aktualnego położenia myszy
   lblX.Text = e.Location.X.ToString();
   lblY.Text = e.Location.Y.ToString();

   /* deklaracje zmiennych pomocniczych i wyznaczenie
       parametrów opisujących prostokąt, w którym będzie kreślona
      figura geometryczna */
int lewyGórnyNarożnikX =
          (Punkt.X > e.Location.X) ? e.Location.X : Punkt.X;
int lewyGórnyNarożnikY =
          (Punkt.Y > e.Location.Y) ? e.Location.Y : Punkt.Y;


 int Szerokość = Math.Abs(Punkt.X - e.Location.X);
 int Wysokość = Math.Abs(Punkt.Y - e.Location.Y);
  /* sprawdzenie, czy obsługiwane zdarzenie  MouseMove
 zostało spowodowane przesunięciem Myszy */
  if (e.Button == MouseButtons.Left)
   {  // rozpoznawanie wybranych (zaznaczonych) kontrolek
         RadioButton, które są „przypisane” odpowiednim
         figurom i liniom geometrycznym
      .    .    .
      .    .    .
      // odświeżenie powierzchni graficznej
      pbRysownica.Refresh();
   }
 } // od pbRysownica_MouseMove
```


```
// rozpoznawanie figur i linii geometrycznych, dla których
chcemy prezentować efekt wizualnego ich rozciągania
if (rdbPunkt.Checked) ; // punktu nie rozciągamy !!!

if (rdbLinia.Checked)   // czy linia prosta
RysownicaTymczasowa.DrawLine(PióroTymczasowe, Punkt.X,
                    Punkt.Y,  e.Location.X, e.Location.Y);
 if (rdbElipsa.Checked)
  RysownicaTymczasowa.DrawEllipse(PióroTymczasowe,
   new Rectangle(lewyGórnyNarożnikX,lewyGórnyNarożnikY,
                        Szerokość, Wysokość));
  if (rdbOkrąg.Checked)
    RysownicaTymczasowa.DrawEllipse(PióroTymczasowe,
  new Rectangle(lewyGórnyNarożnikX, lewyGórnyNarożnikY,
                      Szerokość, Szerokość));

  if (rdbProstokąt.Checked)
    RysownicaTymczasowa.DrawRectangle(PióroTymczasowe,
      new Rectangle(lewyGórnyNarożnikX,
            lewyGórnyNarożnikY, Szerokość, Wysokość));
   if (rdbKwadrat.Checked)
   RysownicaTymczasowa.DrawRectangle(PióroTymczasowe,
   new Rectangle(lewyGórnyNarożnikX,
            lewyGórnyNarożnikY, Szerokość, Szerokość));
   if (rdbLiniaKreślonaMyszą.Checked)
     {   // linii kreślonej myszą „nie rozciągamy”
        Rysownica.DrawLine(Pióro, Punkt.X, Punkt.Y,
                             e.Location.X, e.Location.Y);
       // uaktualnienie początku następnego odcinka linii
       Punkt.X = e.Location.X; Punkt.Y = e.Location.Y;
     }
```