



ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

Departamento de Computación

Computación Gráfica 2283

Introducción

El presente informe documentará el desarrollo de una aplicación en C# que permite realizar cálculos de áreas y perímetros de diversas figuras geométricas mediante el uso de formularios. Esta aplicación, implementada bajo el entorno de desarrollo .NET Framework y utilizando formularios de Windows Forms, tiene como propósito familiarizarse con el entorno de VS así como repasar fundamentos claves de programación orientada a objetos y del uso de varios formularios y subformularios

El proyecto incluye formularios dedicados a cada tipo de figura geométrica, como cuadrados, rectángulos, círculos y triángulos, donde el usuario puede ingresar los parámetros correspondientes, por ejemplo, lados, radio o base y altura. A partir de estos datos, la aplicación realiza los cálculos necesarios y muestra los resultados de manera clara y dinámica. Además, se incorpora un diseño basado en principios de programación estructurada y orientada a objetos, haciendo uso de patrones como el Singleton para optimizar el manejo de ventanas y evitar redundancias en la ejecución.

Requerimientos

Entrada de datos

La entrada de datos para cada formulario está dada por solo los inputs de cada dato que se necesita de la figura y en la parte de las clases se guarda en variables flotantes

Formularios

txtNombreDelInput /*Objeto Tipo TextBox*/

Clases

mNombredelDato /*Objeto Tipo Flotante*/

Salida de datos

La salida de datos está dada por cajas de texto las cuales están inhabilitadas para editar y bloqueadas para poder mover las cajas

Formularios

txtNombredelOutput /*Objeto Tipo TextBox*/

Diseños de la solución

Implementación

Dentro de la implementación de la solución, se encuentran las clases de cada figura, se tiene un constructor público para inicializar las variables de la clase, cada clase tiene su método para recibir los respectivos datos de la figura, asignarlos a las variables, realizar los cálculos del perímetro y del área y el método para escribir dentro del formulario, se añadió la función para dibujar cada figura, en cada uno va a estar en captura los métodos.

Cuadrado

```
1 referencia
public void PlotShape(PictureBox picCanva) {
    using (mGraph = picCanva.CreateGraphics()) {
        mPen = new Pen(Color.Red, 2);

        mGraph.DrawRectangle(mPen, 0, 0, mSide * SF, mSide * SF);
    }
}
```

Triangulo

Todos los métodos que se muestran sirven para poder dibujar el triangulo.

```
2 referencias
public float CalculateSemiPerimeter()...

1 referencia
public void CalculatePerimeter()...

1 referencia
public void CalculateArea()...

1 referencia
public bool CheckTriangle() ...

1 referencia
private void CalculateAngleA()...

1 referencia
private void CalculateVertex()...

1 referencia
public void PlotShape(PictureBox picCanvas)
{
    mGraph = picCanvas.CreateGraphics();
    mPen = new Pen(Color.Blue, 3);
    CalculateVertex();
    mGraph.DrawLine(mPen, A.X * SF, A.Y * SF, B.X * SF, B.Y * SF);
    mGraph.DrawLine(mPen, A.X * SF, A.Y * SF, C.X * SF, C.Y * SF);
    mGraph.DrawLine(mPen, B.X * SF, B.Y * SF, C.X * SF, C.Y * SF);
}
```

Rectángulo

```
1 referencia
public void PlotShape(PictureBox picCanva) {
    using (mGraph = picCanva.CreateGraphics()) {
        mPen = new Pen(Color.Red, 2);

        mGraph.DrawRectangle(mPen, 0, 0, mWidth, mPerimeter);
    }
}
```

Paralelogramo

Se definen puntos para los vértices para luego crear un arreglo de puntos flotantes y luego mandar a dibujar con el método DrawPolygon.

```
1 referencia
public void PlotShape(PictureBox picCanva)
{
    int centerX = picCanva.Width / 2;
    int centerY = picCanva.Height / 2;

    float mBase2 = mBase * SF;
    float mSide2 = mSide * SF;
    float offset = 50.0F;

    PointF A = new PointF(centerX - mBase2 / 2, centerY - mSide2 / 2);
    PointF B = new PointF(centerX + mBase2 / 2, centerY - mSide2 / 2);
    PointF C = new PointF(centerX + mBase2 / 2 + offset, centerY + mSide2 / 2);
    PointF D = new PointF(centerX - mBase2 / 2 + offset, centerY + mSide2 / 2);

    PointF[] parallelogram = { A, B, C, D };

    using (mGraph = picCanva.CreateGraphics()) {
        mPen = new Pen(Color.Red, 2);

        mGraph.DrawPolygon(mPen, parallelogram);
    }
}
```

Rombo

Al igual que la anterior figura se definen los vértices del rombo para mandar a dibujar con el mismo método de la biblioteca Graphics.

```
1 referencia
public void PlotShape(PictureBox picCanva) {
    float centerX = picCanva.Width / 2;
    float centerY = picCanva.Height / 2;

    float mMajorDiagonal2 = mMajorDiagonal * SF;
    float mMinorDiagonal2 = mMinorDiagonal * SF;

    PointF A = new PointF(centerX, centerY - mMajorDiagonal2 / 2);
    PointF B = new PointF(centerX + mMinorDiagonal2 / 2, centerY);
    PointF C = new PointF(centerX, centerY + mMajorDiagonal2 / 2);
    PointF D = new PointF(centerX - mMinorDiagonal2 / 2, centerY);

    PointF[] diamondVertex = {A, B, C, D};

    using (mGraph = picCanva.CreateGraphics()) {
        mPen = new Pen(Color.Red, 2);

        mGraph.DrawPolygon(mPen, diamondVertex);
    }
}
```

Cometa

Con las diagonales y los dos lados más geometría básica se definen los vértices de la cometa para que se dibuje y con ayuda del teorema de Pitágoras se calcula la hipotenusa que sirve como un lado de la cometa

```
public void PlotShape(PictureBox picCanva)
{
    float centerX = picCanva.Width / 2;
    float centerY = picCanva.Height / 2;

    float mMinorDiagonal2 = mMinorDiagonal * SF;
    float mMajorDiagonal2 = mMajorDiagonal * SF;
    float mMajorSide2 = mMajorSide * SF;
    float mMinorSide2 = mMinorSide * SF;

    PointF top = new PointF(centerX, centerY - mMajorDiagonal2 / 2);
    PointF min = new PointF(centerX, centerY + mMajorDiagonal2 / 2);

    float dx = (float)Math.Sqrt(Math.Pow(mMajorSide2, 2) - Math.Pow(mMinorSide2, 2));
    PointF right = new PointF(centerX - dx, centerY);
    PointF left = new PointF(centerX + dx, centerY);

    PointF[] comet = { top, min, right, left };

    using (mGraph = picCanva.CreateGraphics())
    {
        mPen = new Pen(Color.Red, 2);

        mGraph.DrawPolygon(mPen, comet);
    }
}
```

Trapezio

Al igual que las anteriores figuras se definen puntos para los vértices y como nueva variable se tiene el offset que sirve como para desplazar la parte de debajo de la figura para formar el trapezio.

```
public void PlotShape(PictureBox picCanva)
{
    float centerX = picCanva.Width / 2;
    float centerY = picCanva.Height / 2;

    float mMajorBase2 = mMajorBase * SF;
    float mMinorBase2 = mMinorBase * SF;
    float mHeight2 = mHeight + SF;

    float offset = 50;

    PointF A = new PointF(centerX - mMajorBase2 / 2 + offset, centerY - mHeight2 / 2);
    PointF B = new PointF(centerX + mMajorBase2 / 2 + offset, centerY - mHeight2 / 2);
    PointF C = new PointF(centerX + mMinorBase2 / 2, centerY + mHeight2 / 2);
    PointF D = new PointF(centerX - mMinorBase2 / 2, centerY + mHeight2 / 2);

    PointF[] trapeze = { A, B, C, D };

    using (mGraph = picCanva.CreateGraphics()) {
        mPen = new Pen(Color.Red, 2);

        mGraph.DrawPolygon(mPen, trapeze);
    }
}
```

Circulo

Se dibuja normalmente el circulo teniendo en cuenta el dato del radio

```
public void PlotShape(PictureBox picCanva)
{
    using (mGraph = picCanva.CreateGraphics())
    {
        mPen = new Pen(Color.Red, 2);

        mGraph.DrawEllipse(mPen, 0, 0, 2 * mRadius * SF, 2 * SF * mRadius);
    }
}
```

Polígono Regular

Se toma como radio, la longitud del lado del polígono, y el numero de lados se le toma para aplicarlo en el bucle for para definir la figura del polígono regular.

```
public void PlotShape(PictureBox picCanva)
{
    float centerX = picCanva.Width / 2;
    float centerY = picCanva.Height / 2;

    float mRadius = mSide * SF;

    PointF[] point = new PointF[mSideNumber];

    for (int i = 0; i < mSideNumber; i++)
    {
        float angle = (float)(i * 2 * Math.PI / mSideNumber - Math.PI / 2); // Ángulo en radianes
        float x = centerX + mRadius * (float)Math.Cos(angle);
        float y = centerY + mRadius * (float)Math.Sin(angle);
        point[i] = new PointF(x, y);
    }

    using (mGraph = picCanva.CreateGraphics())
    {
        mPen = new Pen(Color.Red, 2);
        mGraph.DrawPolygon(mPen, point);
    }
}
```

Corona Circular.

Se dibujan dos círculos con datos del radio mayor y menor, se rellena el interior de estos dos.

```
public void PlotShape(PictureBox picCanva)
{
    int centerX = picCanva.Width / 2;
    int centerY = picCanva.Height / 2;

    float outerX = centerX - mMajorRadius * SF;
    float outerY = centerY - mMajorRadius * SF;
    float outerDiameter = mMajorRadius * 2 * SF;

    float innerX = centerX - mMinorRadius * SF;
    float innerY = centerY - mMinorRadius * SF;
    float innerDiameter = mMinorRadius * 2 * SF;

    using (mGraph = picCanva.CreateGraphics())
    {
        mPen = new Pen(Color.Red, 2);
        mBrush = new SolidBrush(Color.Gold);

        mGraph.FillEllipse(mBrush, outerX, outerY, outerDiameter, outerDiameter);
        mGraph.FillEllipse(new SolidBrush(Color.Black), innerX, innerY, innerDiameter, innerDiameter);

        mGraph.DrawEllipse(mPen, outerX, outerY, outerDiameter, outerDiameter);
    }
}
```

Sector Circular

Con el for se calcula la parte del sector que debe pintarse dentro del círculo que se dibuja.

```
public void PlotShape(PictureBox picCanva)
{
    float centerX = picCanva.Width / 2;
    float centerY = picCanva.Height / 2;

    float startAngle = 0.0f;

    int steps = 100;

    PointF[] puntos = new PointF[steps + 2];
    puntos[0] = new PointF(centerX, centerY);

    for (int i = 0; i <= steps; i++)
    {
        float angle = startAngle + (float)mGrade / steps * i;
        float radian = angle * (float)Math.PI / 180; // Convertir a radianes

        float x = centerX + mRadius * (float)Math.Cos(radian);
        float y = centerY - mRadius * (float)Math.Sin(radian); // Invertir Y para sistema gráfico
        puntos[i + 1] = new PointF(x, y);
    }

    using (mGraph = picCanva.CreateGraphics())
    {
        Brush sectorBrush = new SolidBrush(Color.Gold);
        mGraph.FillPolygon(sectorBrush, puntos);

        Pen pen = new Pen(Color.Black, 2);
        mGraph.DrawEllipse(pen, centerX - mRadius, centerY - mRadius, mRadius * 2, mRadius * 2);
    }
}
```

Pruebas y Resultados

Después de haber implementado las clases de cada figura en su respectivo formulario, se ingresaron valores en cada figura para comprobar que realizara los cálculos correctos del área y del perímetro, también se ve que los datos se muestran de manera correcta en los textbox correspondientes

Cuadrado

The screenshot shows a window titled 'Figuras' with a sub-form titled 'Cuadrado'. The form is divided into three sections: 'Entradas' (Inputs), 'Procesos' (Processes), and 'Salidas' (Outputs). In the 'Entradas' section, the 'Lado' (Side) is set to 5. The 'Procesos' section contains three buttons: 'Calcular' (Calculate), 'Resetear' (Reset), and 'Regresar' (Back). The 'Salidas' section displays the calculated 'Perimetro' (Perimeter) as 20 and 'Area' as 25. To the right of the input fields is a 'Gráfico' (Graphic) area showing a red square.

Entradas
Lado: 5

Salidas
Perimetro: 20
Area: 25

Triangulo

The screenshot shows a window titled 'Figuras' with a sub-form titled 'Triangulo'. The form is divided into three sections: 'Entradas' (Inputs), 'Procesos' (Processes), and 'Salidas' (Outputs). In the 'Entradas' section, the sides are set to Lado 1: 5, Lado 2: 4, and Lado 3: 7. The 'Procesos' section contains three buttons: 'Calcular' (Calculate), 'Resetear' (Reset), and 'Regresar' (Back). The 'Salidas' section displays the calculated 'Perimetro' (Perimeter) as 16 and 'Area' as 9,797959. To the right of the input fields is a 'Gráfico' (Graphic) area showing a blue triangle.

Entradas
Lado 1: 5
Lado 2: 4
Lado 3: 7

Salidas
Perimetro: 16
Area: 9,797959

Rectángulo

Menu

Figuras

Rectángulo

Entradas

Ancho100

Altura6

Procesos

Calcular

Resetear


Regresar

Salidas

Perimetro212

Area600

Gráfico



Paralelogramo

Menu

Figuras

Paralelogramo

Entradas

Base10

Lado5

Altura2

Procesos

Calcular

Resetear


Regresar

Salidas

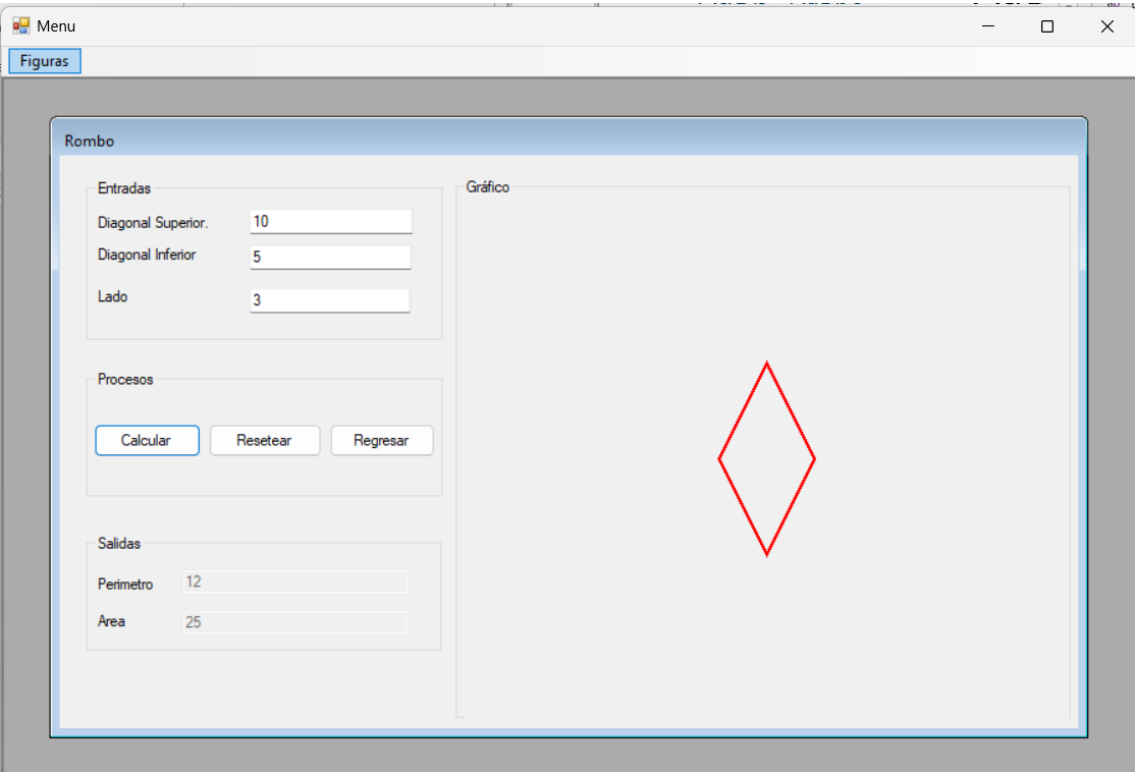
Perimetro20

Area30

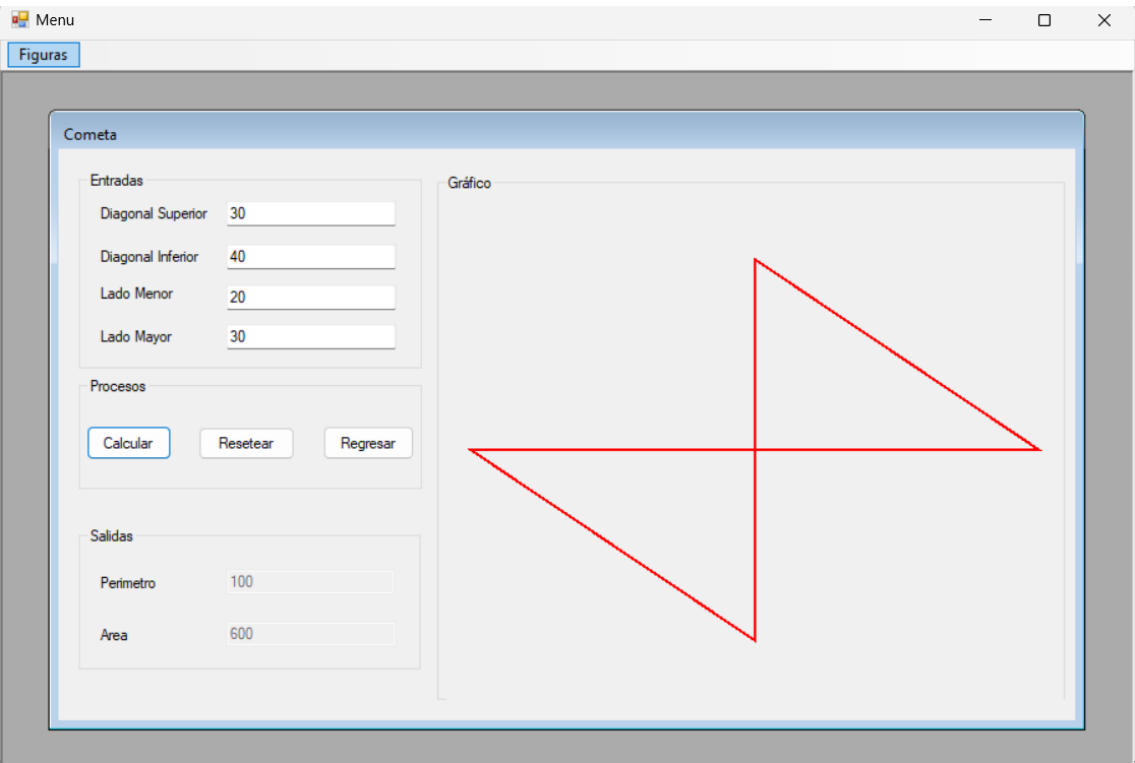
Gráfico



Rombo



Cometa



Trapezio

Menu

Figuras

Trapezio

Entradas

Base Mayor5

Base Menor10

Cateto 17

Cateto 245

Altura5

Procesos

Calcular

Resetear


Regresar

Salidas

Perimetro67

Area37.5

Gráfico



Circulo

Menu

Figuras

Circulo

Entradas

Radio2

Procesos

Calcular

Resetear


Regresar

Salidas

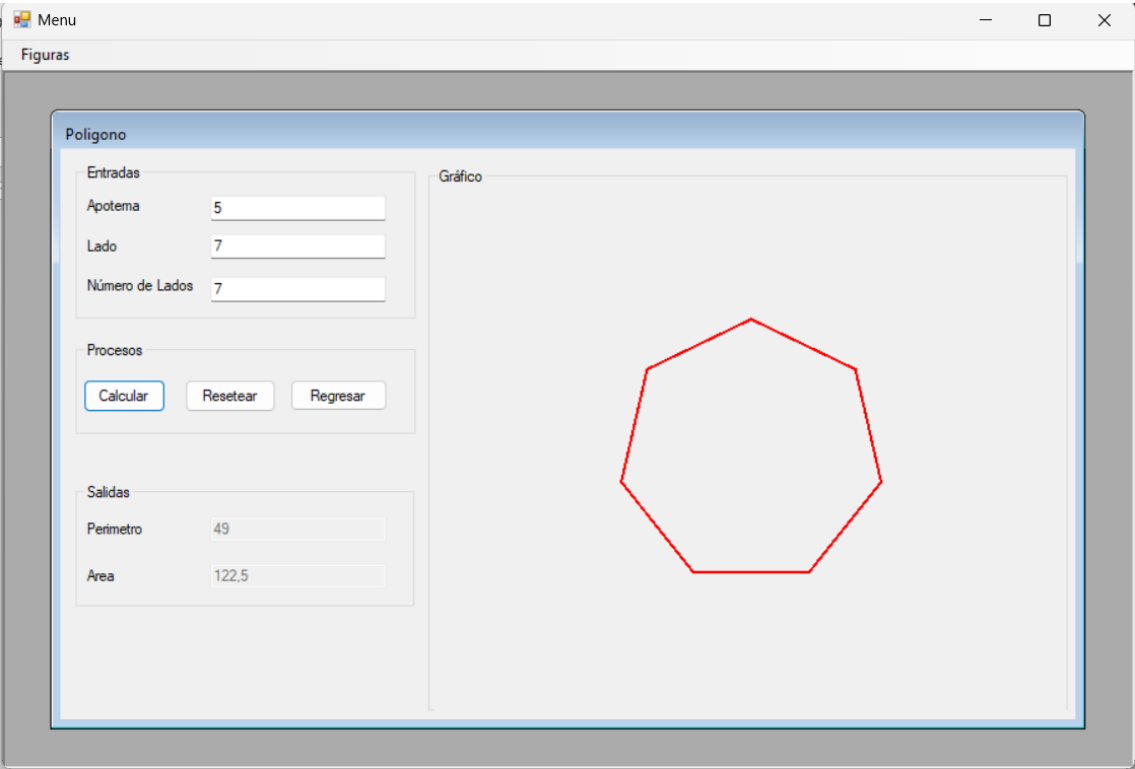
Perimetro12.56637

Area12.56637

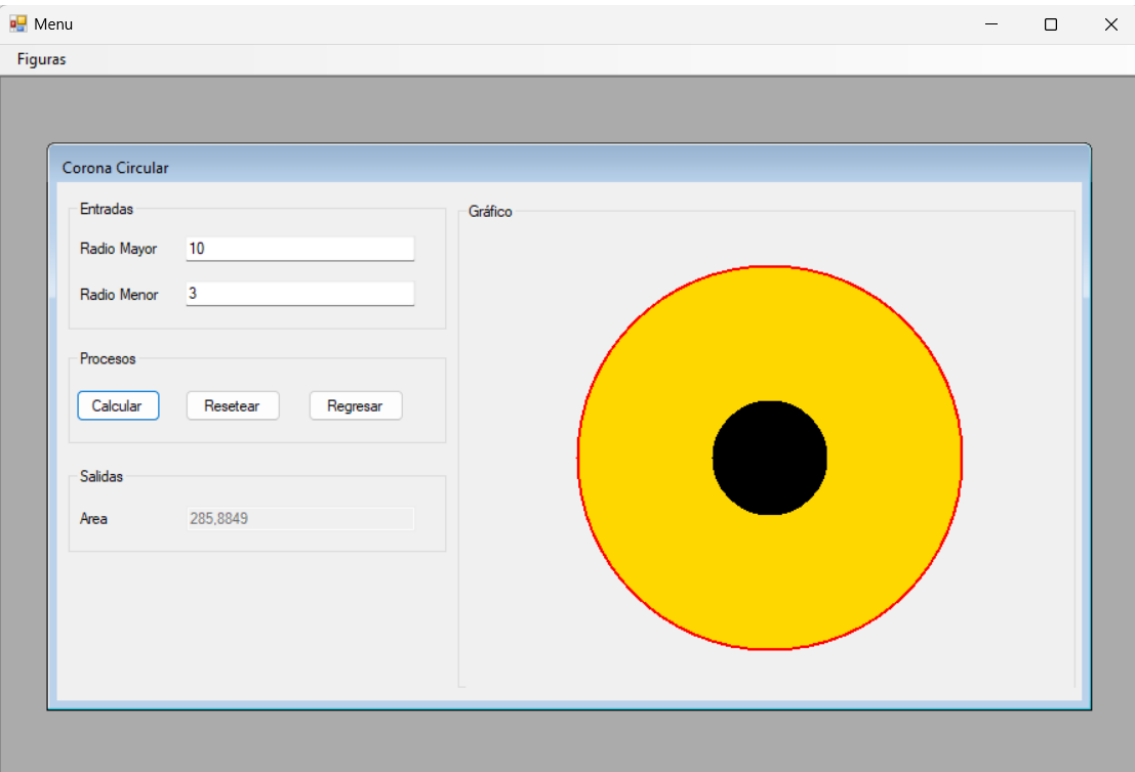
Gráfico



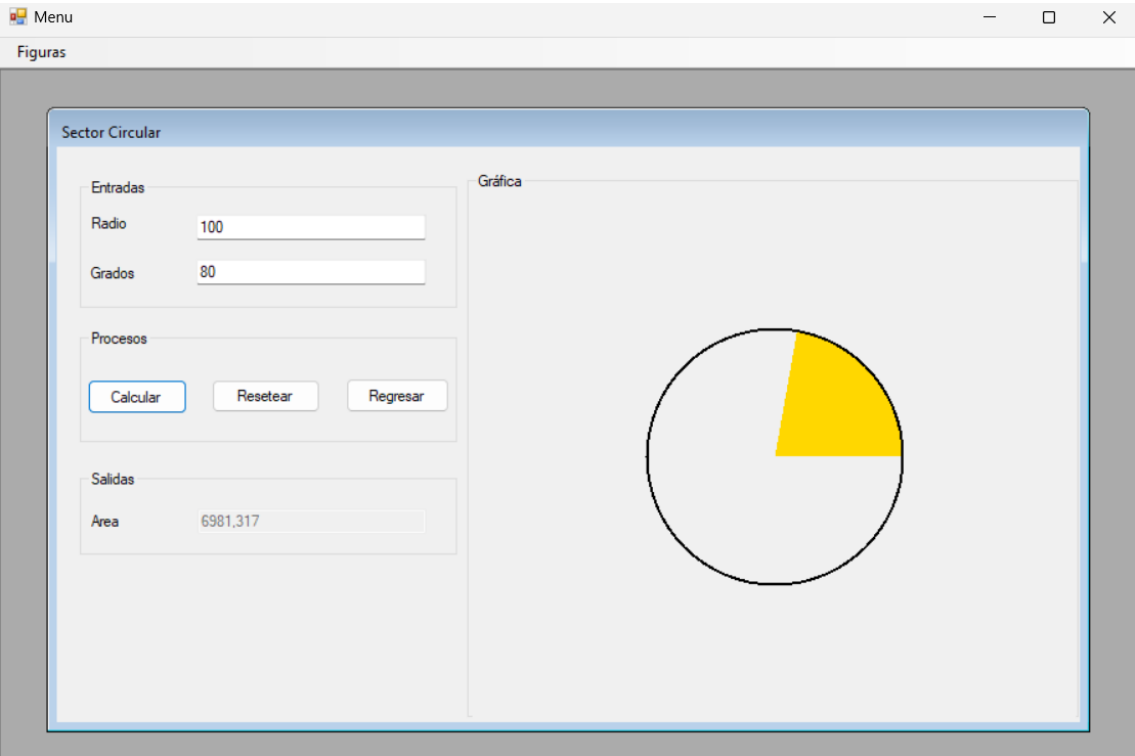
Polígono Regular



Corona Circular



Sector Circular



Conclusiones

- La aplicación desarrollada en C# demostró ser efectiva para realizar cálculos precisos de áreas y perímetros de diversas figuras geométricas, facilitando el aprendizaje y la resolución de problemas matemáticos.
- El uso de formularios de Windows Forms permitió crear una interfaz interactiva y fácil de usar, adecuada tanto para usuarios principiantes como para avanzados. La separación de formularios por figura geométrica simplificó la experiencia de navegación.
- La implementación del patrón Singleton garantizó que solo se abriera una instancia de cada formulario, mejorando el manejo de los recursos y evitando redundancias.

Recomendaciones

- Ampliar la aplicación para incluir cálculos de figuras más complejas como polígonos irregulares, elipses o incluso figuras tridimensionales como prismas y cilindros.
- Asegurarse de que los datos ingresados por los usuarios sean válidos mediante validaciones avanzadas, como restricciones para evitar conflictos con los cálculos, especialmente en figuras como triángulos.
- Manejar la lógica de programación de mejor manera para que a la hora de refactorizarlo no nos de tantos problemas.

Bibliografía

Microsoft. (n.d.). *Windows Forms Overview*. Microsoft Learn. Recuperado el 11 de noviembre de 2024, de <https://learn.microsoft.com/en-us/dotnet/desktop/winforms/>