TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TLAXIACO

CARRERA: INGENIERÍA EN SISTEMAS COMPUTACIONALES

TÓPICOS AVANZADOS DE PROGRAMACIÓN

PRACTICA 5:

PERSONALIZAR UN BOTON

DOCENTE: ING. JOSÉ ALFREDO ROMAN CRUZ

ALUMNO:
Irving Fernando Reyes Pacheco

SEMESTRE:CUARTO **GRUPO:** 4US

ENERO 2022- JULIO 2022

Heroica Ciudad de Tlaxiaco, Oaxaca, a 18 de febrero del 2022.

Practica numero 5

Objetivo:
Realizar el diseño de la interfaz de un menú para una aplicación de escritorio utilizando el programa Visual Studio.

Materiales:

- Una computadora
- Internet
- Visual Studio.

Lista de figuras:

Procedimiento :

Paso 1: Creación de un nuevo proyecto dentro de Visual Studio, en el cual estaremos trabajando la interfaz grafica de nuestro formulario.

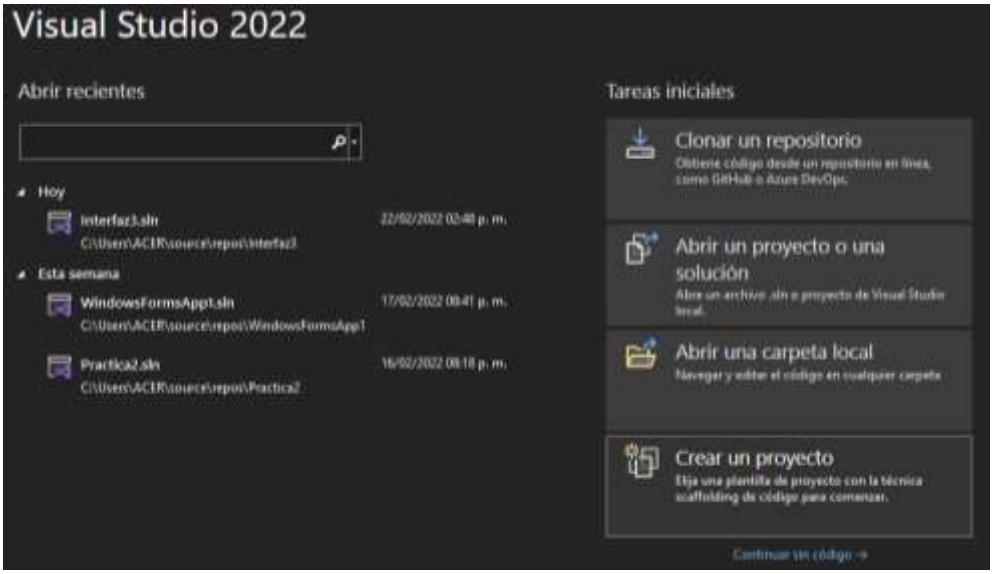

Ilustración 1

Paso 2: Seleccionar el tipo de proyecto que queremos crear, en este caso será una aplicación de Windows Forms

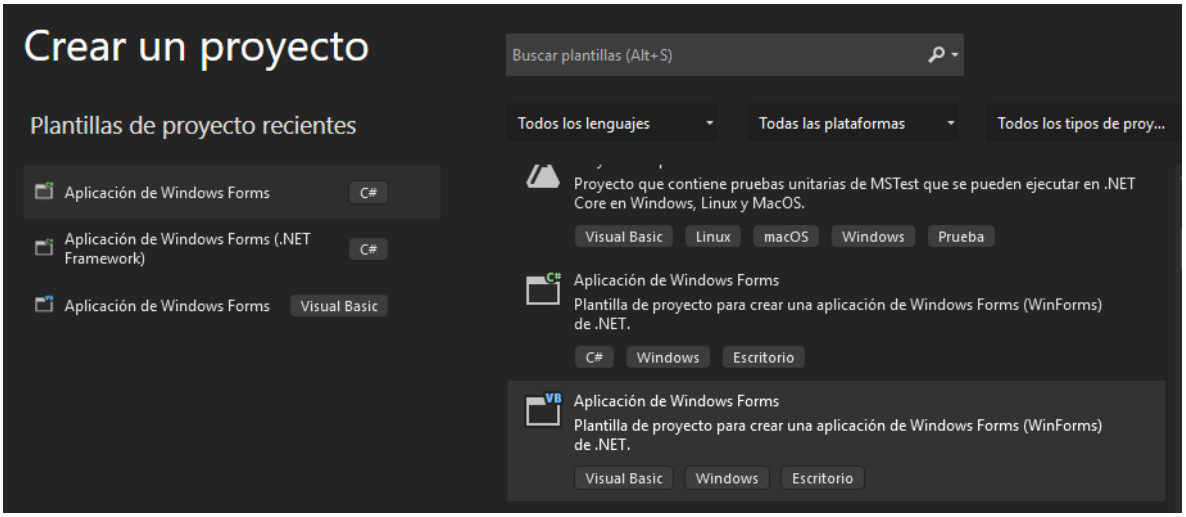

Ilustración 2

Paso 3: Configurar el nuevo proyecto, asignándole el nombre que nosotros queramos.

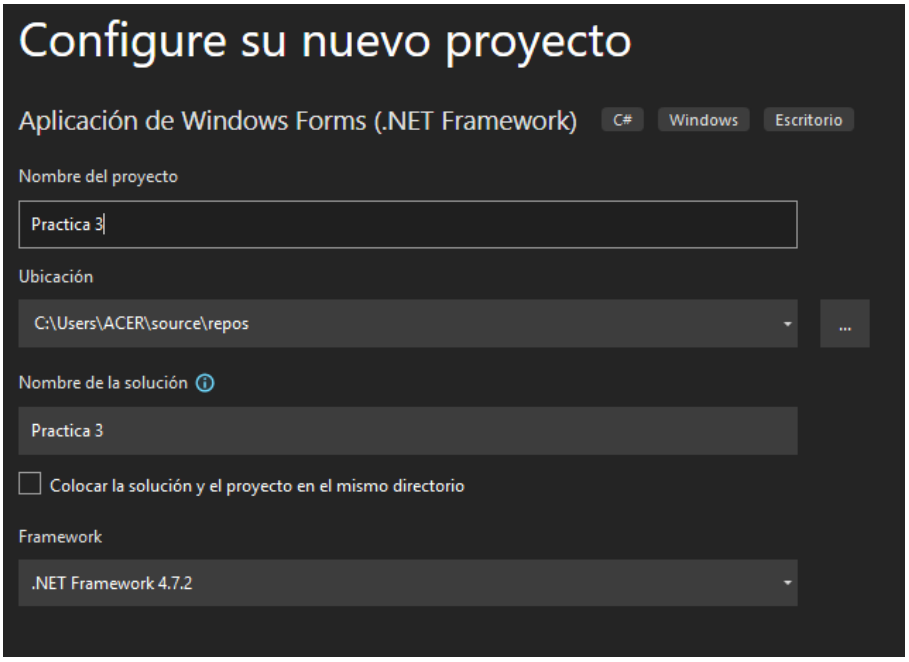

Ilustración 3

4 .-Crear clase
Primeramente, agregaremos una clase para el botón de nombre **personalizar**, en nuestro proyecto de Windows Forms.

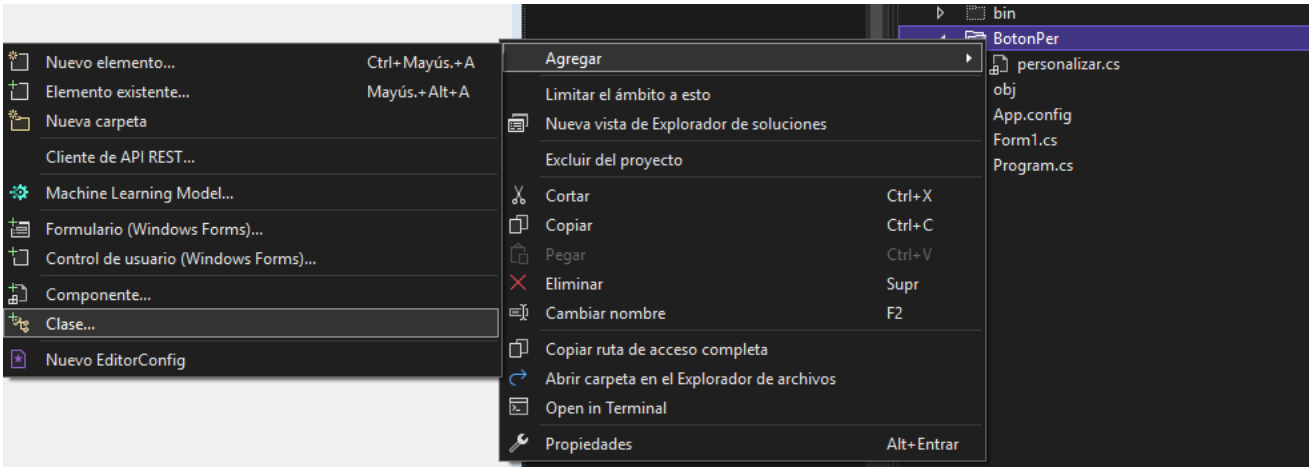

Ilustración 4

5.- Importar librerías

Para construir el botón personalizado, es necesario importar la librería Windows Forms y la librería de dibujos (Drawing) y la librería **ComponentModel** para implementar atributos (Por ejemplo agrupar las propiedades en categorías).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.ComponentModel;
namespace botonpersonalizar.BotonPer
```

Ilustración 5

6.- Heredar la clase Button

Es realmente complicado crear un control personalizado desde cero, por lo que simplemente heredaremos del control Button de la librería Windows Form y así ampliar su funcionalidad y modificar la apariencia.

```
public class RJButton : Button
{
}
```

Ilustración 6

7.- Declarar campos

En la clase, declararemos campos para la apariencia del botón y asignar sus valores predeterminados**,** por ejemplo: El grosor de borde, tamaño del radio de borde y el color de borde.

```
private int borderSize = 0;
private int borderRadius = 20;
private Color borderColor = Color.PaleVioletRed;
```

Ilustración 7

8.- Generar propiedades

Generamos propiedades para exponer los campos anteriores declarados

```
[Category("RJ Code Advance")]
4 referencias
public int BorderSize
{
    get { return borderSize; }
    set
    {
        borderSize = value;
        this.Invalidate();
    }
}
[Category("RJ Code Advance")]
4 referencias
public int BorderRadius
{
    get { return borderRadius; }
    set
    {
        borderRadius = value;
        this.Invalidate();
    }
}
[Category("RJ Code Advance")]
4 referencias
public Color BorderColor
{
    get { return borderColor; }
    set
    {
        borderColor = value;
        this.Invalidate();
    }
```

Ilustración 8

9.- Constructor

En el constructor, inicializaremos algunas propiedades del botón para la apariencia predeterminada. Por ejemplo, un estilo plano y sin bordes, con un tamaño, color de fondo y color de texto específicos. También se puede establecer cualquier otra propiedad que se desee, como un icono, alineación de texto e imagen.

```
4 referencias
public RJButton()
{
    this.FlatStyle = FlatStyle.Flat;
    this.FlatAppearance.BorderSize = 0;
    this.Size = new Size(150, 40);
    this.BackColor = Color.MediumSlateBlue;
    this.ForeColor = Color.White;
    this.Resize += new EventHandler(Button_Resize);
}
```

Ilustración 9

El evento Resize del botón**,** mantiene la proporción del radio del borde limitada a la altura del botón, ya que, si el radio del borde es mayor que la altura del botón, el botón se deformará porque es imposible dibujar un arco más grande en dimensiones más pequeñas.

```
1 referencia
private void Button_Resize(object sender, EventArgs e)
{
    if (borderRadius > this.Height)
        borderRadius = this.Height;
}
```

Ilustración 10

10.- Método crear ruta de figura
Luego de realizar los pasos anteriores, declararemos un método privado para obtener la ruta de gráficos para el diseño del botón con radio de borde personalizable. Para ello:

- Agregaremos un arco en el eje inicial del rectángulo de tamaño igual al radio, comenzando en un ángulo de 180 grados con un rango de recorrido de 90 grados.
- Agregaremos otro arco en la esquina superior derecha, comenzando en un ángulo de 270 grados con un rango de 90 grados.
- Otro arco en la esquina inferior derecha, comenzando en un ángulo de 0 grados con un rango de 90 grados.
- Finalmente agregaremos el ultimo arco en la esquina inferior izquierda para cerrar la forma del botón, comenzando en un ángulo de 90 grados con un rango de 90 grados.

```
2 referencias
private GraphicsPath GetFigurePath(Rectangle rect, float radius)
{
    GraphicsPath path = new GraphicsPath();
    float curveSize = radius * 2F;
    path.StartFigure();
    path.AddArc(rect.X, rect.Y, curveSize, curveSize, 180, 90);
    path.AddArc(rect.Right - curveSize, rect.Y, curveSize, curveSize, 270, 90);
    path.AddArc(rect.Right - curveSize, rect.Bottom - curveSize, curveSize, curveSize, 0, 90);
    path.AddArc(rect.X, rect.Bottom - curveSize, curveSize, curveSize, 90, 90);
    path.CloseFigure();
    return path;
}
```

Ilustración 11

11.- Anular el método OnPaint
Para modificar o ampliar la apariencia del botón es necesario anular el método OnPaint del botón base. En este caso, dibujaremos un botón redondeado o uno tradicional (Cuadrado) dependiendo del valor del radio de borde.

```
protected override void OnPaint(PaintEventArgs pevent)
{
    base.OnPaint(pevent);
    Rectangle rectSurface = this.ClientRectangle;
    Rectangle rectBorder = Rectangle.Inflate(rectSurface, -borderSize, -borderSize);
    int smoothSize = 2;
    if (borderSize > 0)
        smoothSize = borderSize;
    if (borderRadius > 2) //Rounded button
    {
        using (GraphicsPath pathSurface = GetFigurePath(rectSurface, borderRadius))
        using (GraphicsPath pathBorder = GetFigurePath(rectBorder, borderRadius - borderSize))
        using (Pen penSurface = new Pen(this.Parent.BackColor, smoothSize))
        using (Pen penBorder = new Pen(borderColor, borderSize))
        {
            pevent.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
            this.Region = new Region(pathSurface);
            pevent.Graphics.DrawPath(penSurface, pathSurface);
            if (borderSize >= 1)
                pevent.Graphics.DrawPath(penBorder, pathBorder);
        }
    }
    else
    {
        pevent.Graphics.SmoothingMode = SmoothingMode.None;
        this.Region = new Region(rectSurface);
        if (borderSize >= 1)
        {
            using (Pen penBorder = new Pen(borderColor, borderSize))
            {
                penBorder.Alignment = PenAlignment.Inset;
                pevent.Graphics.DrawRectangle(penBorder, 0, 0, this.Width - 1, this.Height - 1);
```

Ilustración 12

12.- Anular el método OnHandleCreated

Tambien es necesario anular el método OnHandleCreated. Usaremos este método para solucionar el siguiente inconveniente: En el método anterior, para tener un botón de buena calidad de imagen, se dibuja el borde de la superficie con el mismo color de su padre contenedor. Por lo tanto, si el formulario o control contenedor cambia de color de fondo, el borde de la superficie del botón se hará visible dejando una mala impresión.

```
0 referencias
protected override void OnHandleCreated(EventArgs e)
{
    base.OnHandleCreated(e);
    this.Parent.BackColorChanged += new EventHandler(Container_BackColorChanged);
}
1 referencia
private void Container_BackColorChanged(object sender, EventArgs e)
{
    this.Invalidate();
}
```

Ilustración 13

Selección del botón que hemos creado

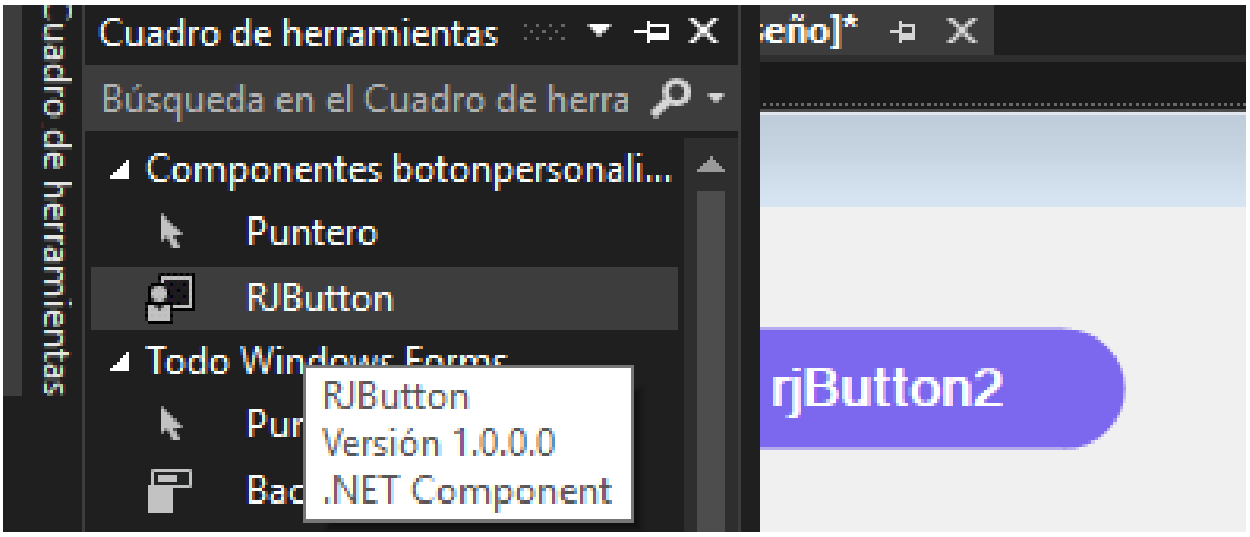

Ilustración 14

Propiedades del botón que se ha creado, donde podemos modificar sus atributos para que muestre un diseño diferente.

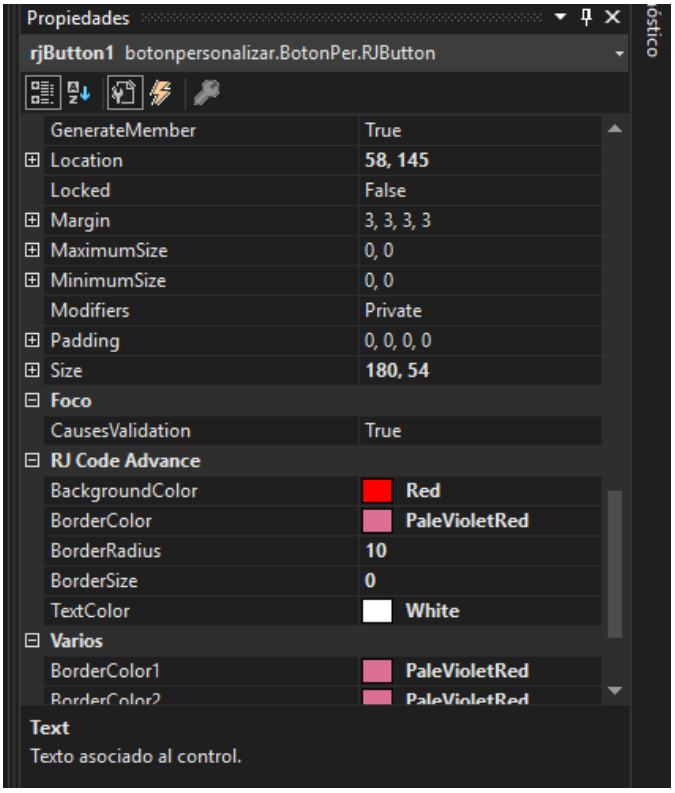

Ilustración 15

Aplicación de escritorio con el botón personalizado terminada antes de ser ejecutada

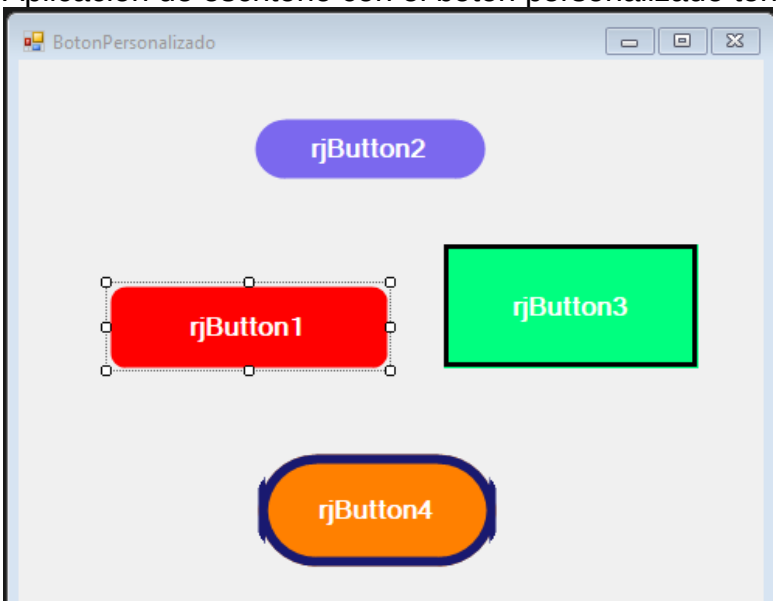

Ilustración 16

Aplicación de escritorio con nuestro botón modificado después de ser ejecutado

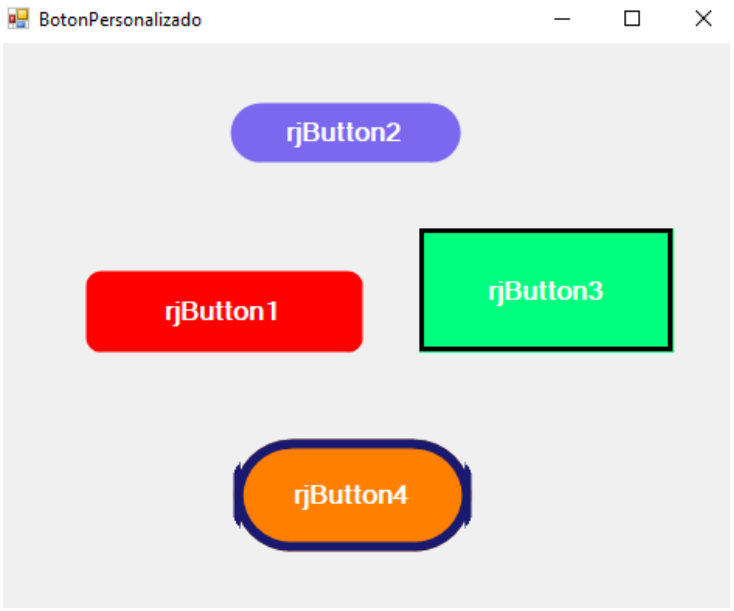

Ilustración 17

Conclusión

Después de realizar esta practica puedo decir que Visual Studio es un programa muy completo en cuanto a lo que son los diferentes tipos de proyectos que podemos realizar en él, sin duda ofrece una gran cantidad de herramientas que permiten el diseño de un formulario, así como una gran cantidad de propiedades para hacer distintas modificaciones, esto permite darle un gran diseño a cualquier proyecto, además de que posee una gran cantidad de comandos para el desarrollo de interfaz con un desarrollo muy técnico y profesional y aunque es verdad que este programa posee muchos comandos y atributos también podemos modificarlos para crear algunos propios, estos comandos mas personalizados los podemos comprar dentro de los mismos programas, sin embargo también podemos crearlos, de modo que sean como nosotros queremos, esto lo podemos hacer modificando las propiedades y atributos de algún comando que queramos modificar, en esos casos podemos aplicar algo a lo que se le conoce como herencia, lo cual nos permite heredar las propiedades de un comando base y modificarlas o mejorarlas tanto como queramos.