TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

Desarrollo de una interfaz gráfica para una Calculadora Estándar utilizando Eventos C#.

Presenta:
Equipo YBJS:
Brisa Osorio Ramírez.
Jazmín López Riveros.
Sandra Yolotzin Reyes García.
Yabin España Aparicio.

Grupo:
4US.

Semestre:
4°.

Asignatura:
Tópicos Avanzados de Programación SCD-1027.

Carrera:
Ingeniera en Sistemas Computacionales – Desarrollo de Software.
ISIE-DES-2019-01

Producto:
Reporte de Práctica.

Profesor:
Ing. José Alfredo Román Cruz.

Tlaxiaco, Oaxaca, 16 de marzo de 2021

Contenido

1. Presentación.

En esta práctica se presenta el e procedimiento que se lleva acabo para poder desarrollar una calculadora con operaciones básicas en Visual Studio,.

2. Generalidades de la Práctica.

2.1 Objetivos.

- Diseñar una calculadora con operaciones básicas.
- Establecer buen orden en el código para que el lector pueda visualizar fácilmente el desarrollo gráfico de la aplicación.
- Utilizar POO y eventos en la codificación.

El diseño plano (en inglés, flat design) es un tipo de diseño de interfaz de usuario minimalista o lengua de diseño, generalmente utilizado en interfaces de usuario gráfico (como aplicaciones web y aplicaciones móviles), especialmente en material gráfico como pósteres, banners, artes, documentos de guía y publicidad de productos. (Wikipedia, 2021)

2.1 Material.

- Computadora Personal.
- Visual Studio.
- Internet.
- Recursos; como imágenes y vectores.

3. Procedimiento.

3.1 Creación del Proyecto.

1. Creamos un nuevo proyecto dando Archivo.Nuevo.Proyecto.

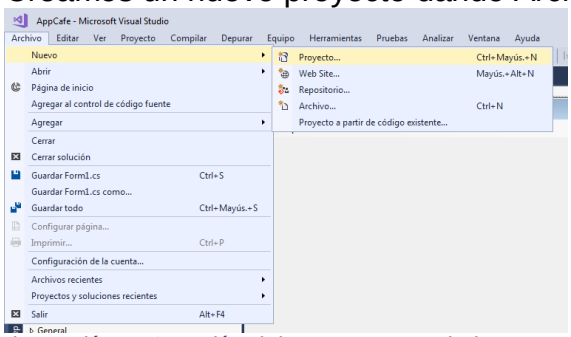


Ilustración 1. Creación del Proyecto en Windows Forms

2. Seleccionamos Aplicación de Windows Forms (.Net Framework).

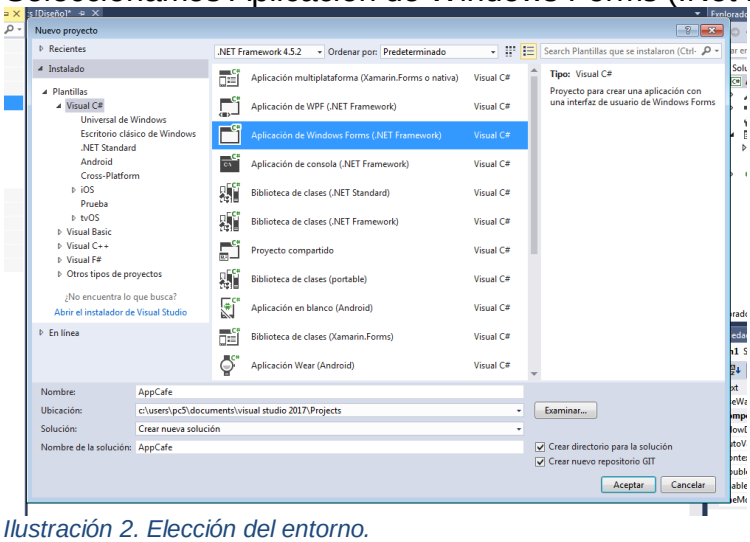


Ilustración 2. Elección del entorno.

3. Luego de crear el Proyecto la interfaz de diseño será la siguiente:

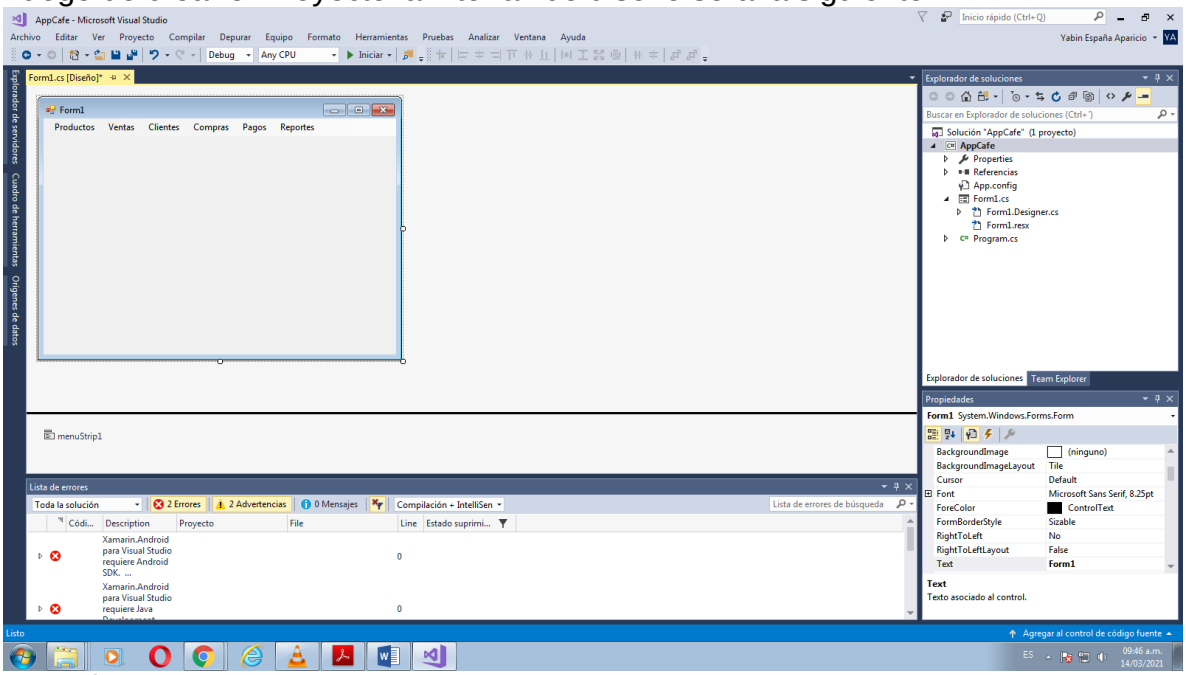


Ilustración 3. Entorno en Forms.

4. Para conocer las herramientas de Visual Studio en Windows Forms tenemos;

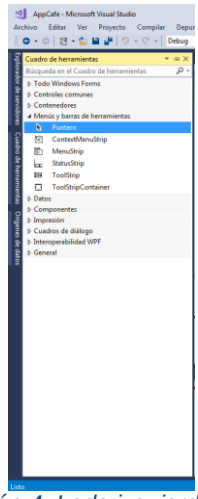


Ilustración 4. Lado izquierdo del entorno.

Del lado izquierdo el Cuadro de herramientas y del lado derecho las Propiedades y el explorador de Soluciones.

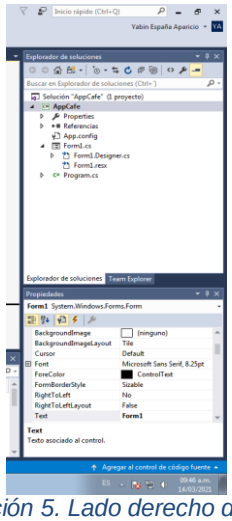
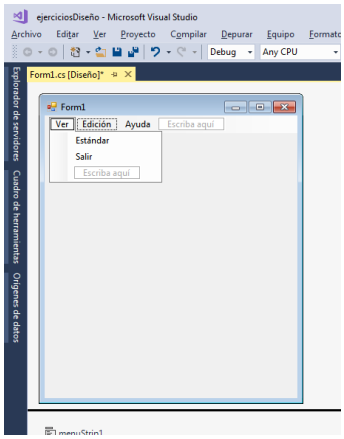


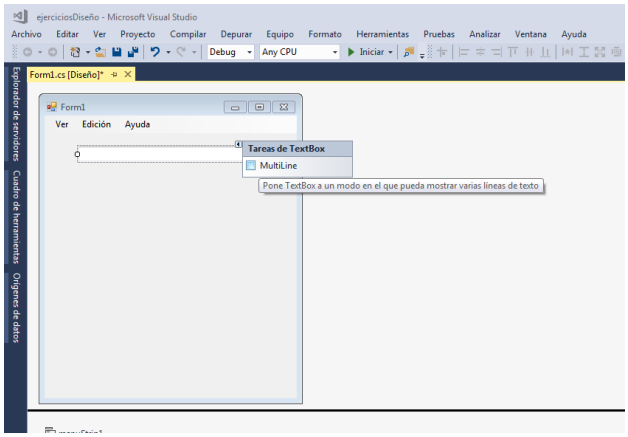
Ilustración 5. Lado derecho del entorno.

3.2 Inserción de Elementos.

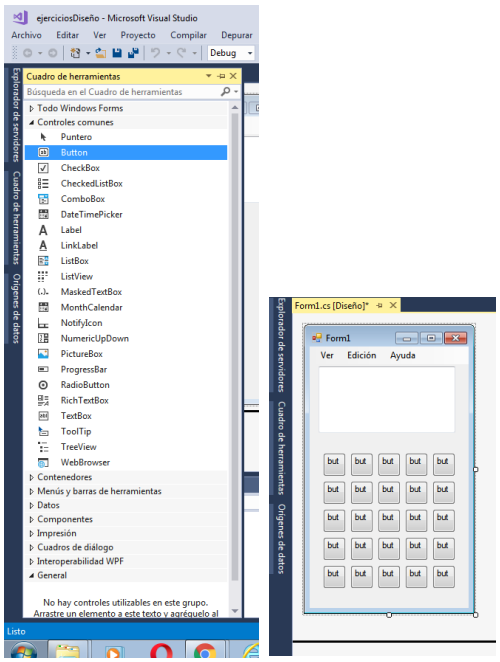
5. Una vez dentro del entorno insertamos un menú mediante MenuSrip:



6. Insertamos un TextBox:

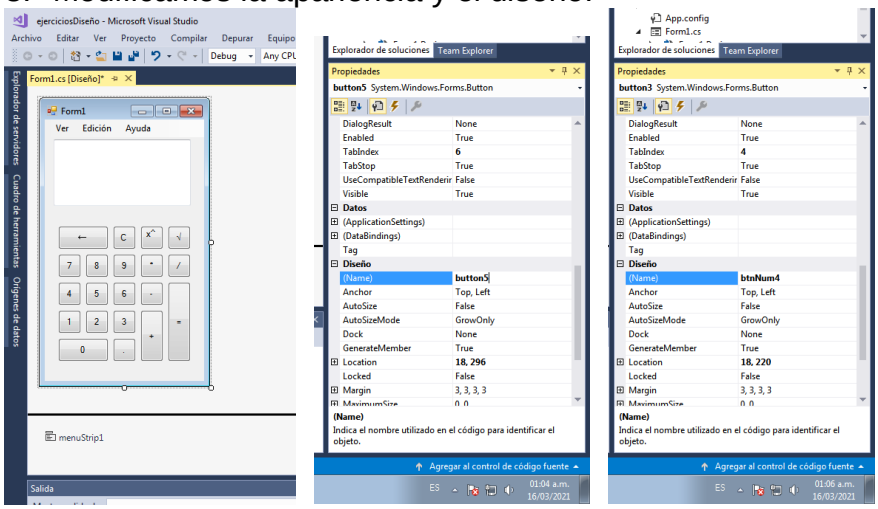


7. Insertamos Botones.



3.2.1 Nombres y etiquetas.

8. Modificamos la apariencia y el diseño.[1]



3.3 Métodos del Formulario.

3.3.1 Métodos del comportamiento de los botones.

9. Ahora procederemos a crear eventos en cada button.

```
private void btnNum1_Click(object sender, EventArgs e)
{

}

private void btnNum2_Click(object sender, EventArgs e)
{

}

private void btnNum3_Click(object sender, EventArgs e)
{

    }
```

10. Establecemos una variable hará la sucesión cronológica del form:

```
Boolean secuencia = true;
```

11. Una vez dentro de los eventos, se escriben los comportamientos.

```
if(secuencia == true)
   {
      txtBoxVentana.Text = "";
      txtBoxVentana.Text = "1";
      secuencia = false;
   }
   else
   {
      txtBoxVentana.Text = txtBoxVentana.Text + "1";
   }
```

12. Se realizan los comportamientos en cada evento de manera que:

[1] Modificando en la parte del diseño el nombre con prefijos como btn < buttons o txtBox < textBox y en la apariencia el text, lo que se va a ver en el programa. Esto para tener un orden y no generar confusión al momento de programar.

```
private void btnNum1_Click(object sender, EventArgs e)
{
   if(secuencia == true)
   {
      txtBoxVentana.Text = "";
      txtBoxVentana.Text = "1";
      secuencia = false;
   }
   else
   {
      txtBoxVentana.Text = txtBoxVentana.Text + "1";
   }
}

private void btnNum2_Click(object sender, EventArgs e)
{
   if (secuencia == true)
   {
      txtBoxVentana.Text = "";
      txtBoxVentana.Text = "2";
      secuencia = false;
   }
   else
   {
      txtBoxVentana.Text = txtBoxVentana.Text + "2";
   }
}
```

3.3.2 Métodos de Operadores Simples:

13. Establecemos las variables a utilizar en los operadores:

```
double variable1,
```

14. Insertamos el comportamiento de los operadores:

```
private void btnSuma_Click(object sender, EventArgs e)
{
   operacion = "+";
   variable1 = double.Parse(txtBoxVentana.Text);
   secuencia = true;
}
private void btnResta_Click(object sender, EventArgs e)
{
   operacion = "-";
   variable1 = double.Parse(txtBoxVentana.Text);
   secuencia = true;
}
private void btnMultiplicacion_Click(object sender, EventArgs e)
{
   operacion = "*";
   variable1 = double.Parse(txtBoxVentana.Text);
   secuencia = true;
}

private void btnDivision_Click(object sender, EventArgs e)
```

```
{
    operacion = "/";
    variable1 = double.Parse(txtBoxVentana.Text);
    secuencia = true;

}
```

15. Establecemos variables para los siguientes comportamientos:

```
    string operacion, borrado;

double variable2, resultado;
```

16. Los comportamientos Radicación y Exponenciación son diferentes a los otros operadores y los definimos mediante:

```
private void btnExponenciacion_Click(object sender, EventArgs e)
{
    variable1 = double.Parse(txtBoxVentana.Text);
    resultado = variable1 * variable1;
    txtBoxVentana.Text = resultado.ToString();
    secuencia = true;
}

private void btnRadicacion_Click(object sender, EventArgs e)
{
    variable1 = double.Parse(txtBoxVentana.Text);
    resultado = Math.Sqrt(variable1);
    txtBoxVentana.Text = resultado.ToString();
    secuencia = true;

    }
```

17. Establecemos el operador igual mediante:

```
private void btnIgual_Click(object sender, EventArgs e)
{
    variable2 = double.Parse(txtBoxVentana.Text);
    if (operacion == "+")
    {
        resultado = variable1 + variable2;
        txtBoxVentana.Text = resultado.ToString();
        secuencia = true;
    }
    if (operacion == "-")
    {
        resultado = variable1 - variable2;
        txtBoxVentana.Text = resultado.ToString();
        secuencia = true;
    }
    if (operacion == "*")
    {
        resultado = variable1 * variable2;
        txtBoxVentana.Text = resultado.ToString();
        secuencia = true;
    }
    if (operacion == "/")
```

```
        {
            resultado = variable1 / variable2;
            txtBoxVentana.Text = resultado.ToString();
            secuencia = true;
        }
    }
```

3.3.3 Métodos de Operadores Especiales:

18. Definimos comportamientos especiales mediante:

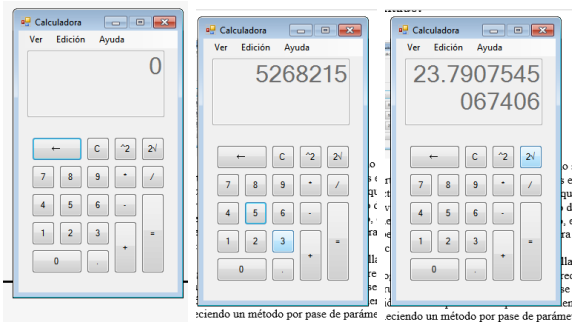
```
private void btnClearTextBox_Click(object sender, EventArgs e)
{
    txtBoxVentana.Text = "0";
    variable1 = 0;
    variable2 = 0;
    secuencia = true;
}

private void btnRetroceso_Click(object sender, EventArgs e)
{
    int x;
    borrado = txtBoxVentana.Text;
    x = borrado.Length - 1;
    borrado = borrado.Substring(0, x);
    txtBoxVentana.Text = borrado;

    if (txtBoxVentana.Text == "")
    {
        txtBoxVentana.Text = "0";
        secuencia = true;
    }
    if(txtBoxVentana.Text== "-")
    {
        txtBoxVentana.Text = "0";
        secuencia = true;
    }
}
private void salirToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}
```

3.4Resultados y Conclusiones.

Resultado:



En conclusión, el diseño plano como se pudo apreciar, utiliza muy pocos elementos. La particularidad que tiene, son los eventos en el código, estos tienen que estar muy bien estructurados para poder funcionar, por lo que utilizan la modulación con respecto a, subdividir las operaciones. En el desarrollo de aplicaciones se utilizan más los métodos ya establecidos por el programa Visual Studio, en futuros desarrollos recomiendo más utilizar las operaciones de Bibliotecas de clases para así especificar tu propio comportamiento de operaciones.

Las funcionalidades se irán desarrollando conforme el curso de Tópicos Avanzados de Programación. En el código se logró apreciar el uso de la estructura de datos mediante el constructor generado por Default, además se utilizaron otros métodos que llaman la atención de cómo puedes manipular fácilmente el comportamiento de los Buttons estableciendo un método por pase de parámetro por valor.