Fundamentos

Temas a tratar:

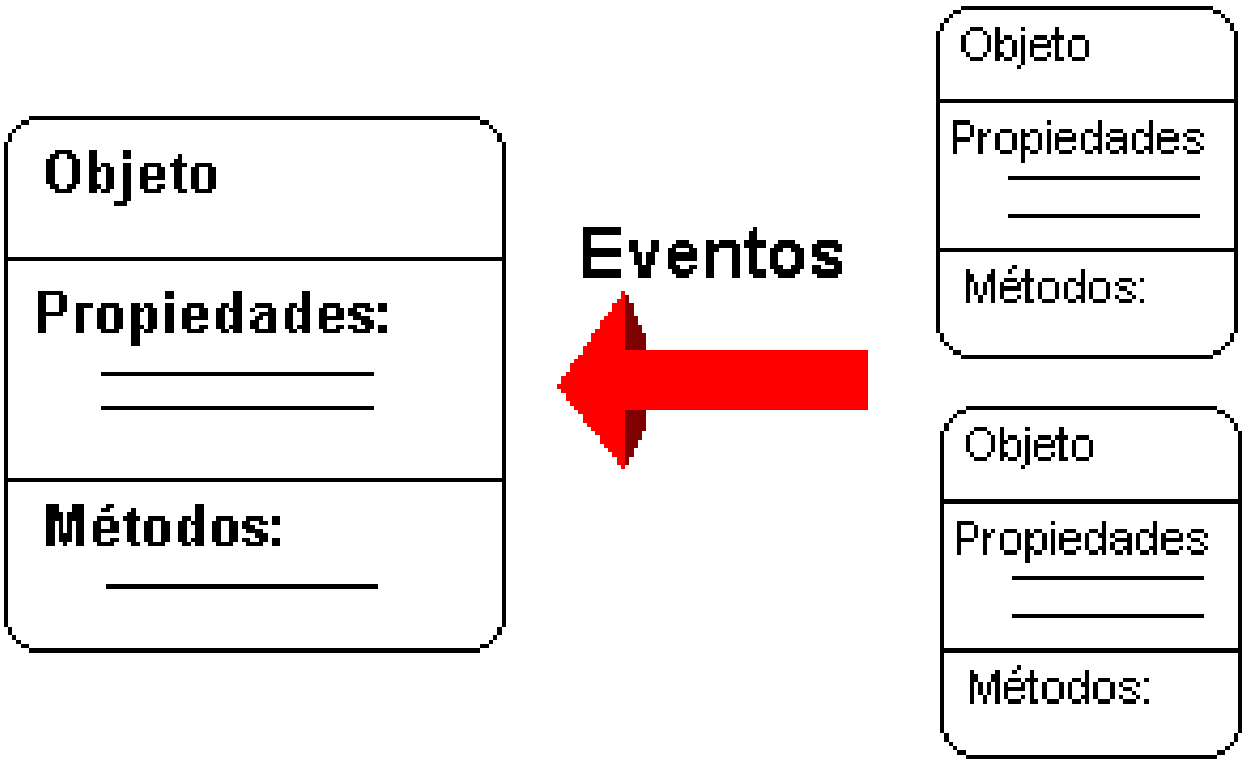
2.1.- Programación Orientada a Objetos

Todos los lenguajes administrados de .NET Framework, como Visual Basic y C#, proporcionan plena compatibilidad con la programación orientada a objetos, incluidos la encapsulación, la herencia y el polimorfismo.

La **encapsulación** significa que un grupo de propiedades, métodos y otros miembros relacionados se tratan como si de una sola unidad u objeto se tratase.

Herencia describe la posibilidad de crear nuevas clases basadas en una clase existente.

Polimorfismo significa que puede tener múltiples clases que se pueden utilizar de forma intercambiable, si bien cada clase implementa las mismas propiedades o los mismos métodos de maneras diferentes.



2.2.- Clase

Los términos **clase y objeto** se usan a veces indistintamente pero, en realidad, las clases describen el **tipo** de los objetos, mientras que los objetos son **instancias** de clases que se pueden usar. Así, la acción de crear un objeto se denomina **creación de instancias**. Con la analogía de plano, una clase es un plano y un objeto es un edificio construido a partir de ese plano.

Para definir una clase:

```
class SampleClass {}
```

2.3.- Estructura

Tanto Visual Basic como C# también proporcionan una versión ligera de las clases denominadas **estructuras**, que resultan útiles cuando es necesario crear una matriz grande de objetos y no se desea usar demasiada memoria para ello.

Para definir una estructura:

```
struct SampleStruct { }
```

Cada clase puede tener distintos miembros de clase, entre los que se incluyen las propiedades que describen los datos de clase, los métodos que definen el comportamiento de la clase y los eventos que proporcionan comunicación entre distintos objetos y clases.

Propiedades y campos

Los campos y propiedades representan información que contiene un objeto.
Los campos se parecen a las variables ya que se pueden leer y establecer directamente.

Para definir un campo:

```
Class SampleClass
{
        public string sampleField;
}
```

Propiedades tienen procedimientos get y set , que proporcionan más control sobre cómo se establecen o devuelven los valores.

Tanto C# como Visual Basic permiten crear un campo privado para almacenar el valor de propiedad o bien usar las denominadas propiedades de implementación automática que crean este campo en segundo plano automáticamente y proporcionan la lógica básica para los procedimientos de propiedad.

Para definir una propiedad implementada automáticamente:

```
class SampleClass
{
        public int SampleProperty { get; set; }
}
```

Si necesita realizar algunas operaciones adicionales para leer y escribir el valor de propiedad, defina un campo para almacenar el valor de propiedad y proporcione la lógica básica para almacenarlo y recuperarlo:

```
class SampleClass
{
        private int _sample;
        public int Sample
        {
                // Retorna el valor almacenado del campo.
                get { return _sample; }
                // Almacena el valor en el campo.
                set { _sample = value; }
        }
}
```

Método

Un método es una acción que puede realizar un objeto.

Para definir un método de una clase:

```
class SampleClass
{
        public int sampleMethod(string sampleParam)
        {
                // Insert code here
        }
}
```

Una clase puede tener varias implementaciones o **sobrecargas** del mismo método que se diferencian en el número de parámetros o de tipos de parámetro.
Para sobrecargar un método:

```
public int sampleMethod(string sampleParam) {};
public int sampleMethod(int sampleParam) {}
```

En la mayoría de los casos, un método se declara dentro de una definición de clase. Sin embargo, tanto Visual Basic como C# también admiten los **métodos de extensión**, que permiten agregar métodos a una clase existente fuera de la definición de la clase en sí.

Constructores

Los constructores son métodos de clase que se ejecutan automáticamente cuando se crea un objeto de un tipo determinado. Normalmente, los constructores inicializan los miembros de datos del nuevo objeto. Un constructor se puede ejecutar sólo una vez cuando se crea una clase.

Además, el código del constructor siempre se ejecuta antes que cualquier otro código en una clase. Sin embargo, puede crear varias sobrecargas del constructor de la misma forma que para cualquier otro método.

Para definir un constructor para una clase:

```
public class SampleClass
{
        public SampleClass() {
                // Add code here
        }
}
```

Destructores

Los destructores se utilizan para destruir instancias de clases. En .NET Framework, el recolector de elementos no utilizados administra automáticamente la asignación y liberación de memoria para los objetos administrados en la aplicación. Sin embargo, puede que aún necesite destructores para limpiar recursos no administrados que crea la aplicación. Sólo puede haber un destructor para cada clase.

Eventos

Los eventos habilitan que una clase u objeto notifique a otras clases u objetos cuando se produce algo interesante . La clase que envía (o genera) el evento recibe el nombre de publicador y las clases que reciben (o controlan) el evento se denominan suscriptores.

Modificadores y niveles de acceso

Modificador de Visual Basic	Modificador de C#	Definición
Public (Visual Basic)	public	Puede obtener acceso al tipo o miembro cualquier otro código del mismo ensamblado o de otro ensamblado que haga referencia a éste.
Private (Visual Basic)	private	Solamente puede obtener acceso al tipo o miembro el código de la misma clase.
Protected (Visual Basic)	protected	Solamente puede obtener acceso al tipo o miembro el código de la misma clase o de una clase derivada.
Friend (Visual Basic)	internal	Puede obtener acceso al tipo o miembro cualquier código del mismo ensamblado, pero no de un ensamblado distinto.
Protected Friend	**protected internal**	Puede obtener acceso al tipo o miembro cualquier código del mismo ensamblado o cualquier clase derivada de otro ensamblado.

Creación de instancias de las clases

Para crear un objeto, debe crear una o varias instancias de una clase.

```
SampleClass sampleObject = new SampleClass();
```

Una vez creadas las instancias de una clase, puede asignar valores a las propiedades y los campos de la instancia, así como invocar métodos de clase.

```
// Set a property value.
sampleObject.sampleProperty = "Sample String";
// Call a method.
sampleObject.sampleMethod();
```

Para asignar valores a las propiedades durante el proceso de creación de instancias de una clase, use los inicializadores de objeto:

```
// Set a property value.
SampleClass sampleObject = new SampleClass
        { FirstProperty = "A", SecondProperty = "B" };
```

Clases y miembros estáticos (compartidos)

Un miembro estático (compartido en Visual Basic) de la clase es una propiedad, un procedimiento o un campo que comparten todas las instancias de una clase.
Para definir un miembro estático (compartido):

```
static class SampleClass {
        public static string SampleString = "Sample String";
}
```

Para obtener acceso al miembro estático (compartido), use el nombre de la clase sin crear un objeto perteneciente a esta:

```
Console.WriteLine(SampleClass.SampleString);
```

Las clases estáticas (compartidas) de C# y los módulos de Visual Basic solamente tienen miembros estáticos (compartidos) y no se pueden crear instancias de los mismos. Además, los miembros estáticos (compartidos) tampoco pueden tener acceso a las propiedades, los campos o los métodos no estáticos (no compartidos).

Tipos anónimos

Los tipos anónimos permiten crear objetos sin escribir una definición de clase para el tipo de datos. En su lugar, el compilador genera una clase. La clase no tiene ningún nombre que se pueda usar y contiene las propiedades especificadas al declarar el objeto.
Para crear una instancia de un tipo anónimo:

```
// sampleObject is an instance of a simple anonymous type.
var sampleObject = new {
        FirstProperty = "A", SecondProperty = "B" };
```

Herencia

La herencia permite crear una nueva clase que reutiliza, extiende y modifica el comportamiento que se define en otra clase. La clase cuyos miembros se heredan se denomina *clase base* y la clase que hereda esos miembros se denomina *clase derivada*. Sin embargo, todas las clases de C# y Visual Basic heredan implícitamente de la clase **Object** que admite la jerarquía de clases .NET y proporciona servicios de bajo nivel a todas las clases.

```
class DerivedClass:BaseClass{}
```

De forma predeterminada, todas las clases se pueden heredar. Sin embargo, puede especificar si una clase no se debe usar como clase base o bien crear una clase que solo se pueda usar como clase base. Para especificar que una clase no se puede usar como clase base:

```
public sealed class A { }
```

Para especificar que una clase se puede usar solo como clase base y no se pueden crear instancias de esta:

```
public abstract class B { }
```

Las interfaces, como las clases, definen un conjunto de propiedades, métodos y eventos. Pero a diferencia de las clases, las interfaces no proporcionan implementación. Se implementan por las clases y se definen como entidades separadas desde las clases. Una interfaz representa un contrato, en el cual una clase que implementa una interfaz debe implementar cualquier aspecto de dicha interfaz exactamente como esté definido.

Para definir una interfaz:

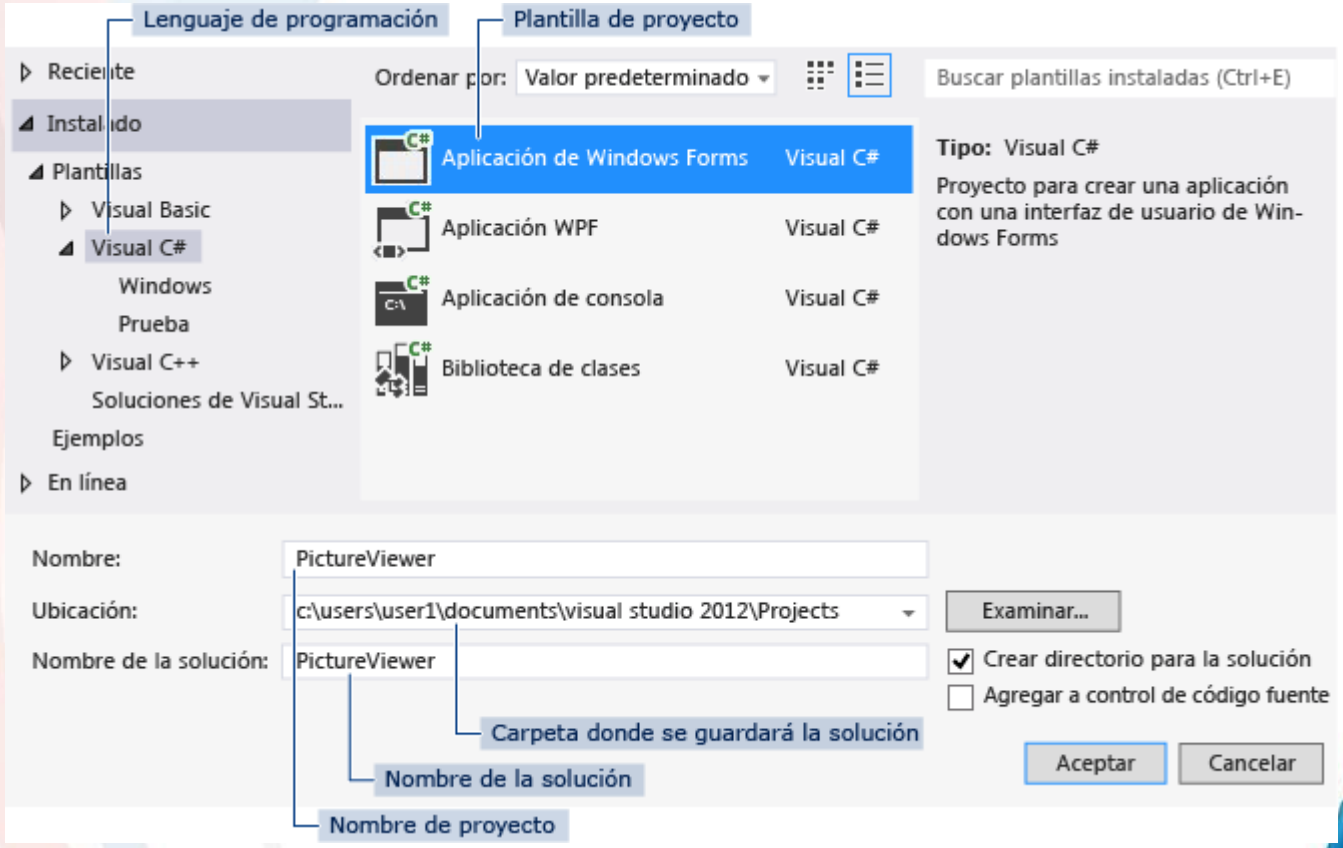
```
interface ISampleInterface
{
        void doSomething();
}
```

Para implementar una interfaz en una clase:

```
class SampleClass : ISampleInterface
{
        void ISampleInterface.doSomething()
        {
                // Method implementation.
        }
}
```

2.4.- Formularios

Crear un proyecto de Aplicación de Windows Forms



En la barra de menús, elija **Archivo**, **Nuevo**, **Proyecto**. El cuadro de diálogo debe tener un aspecto similar al que se muestra a continuación.

1.Elija **Visual C#** o **Visual Basic** en la lista **Plantillas instaladas**.

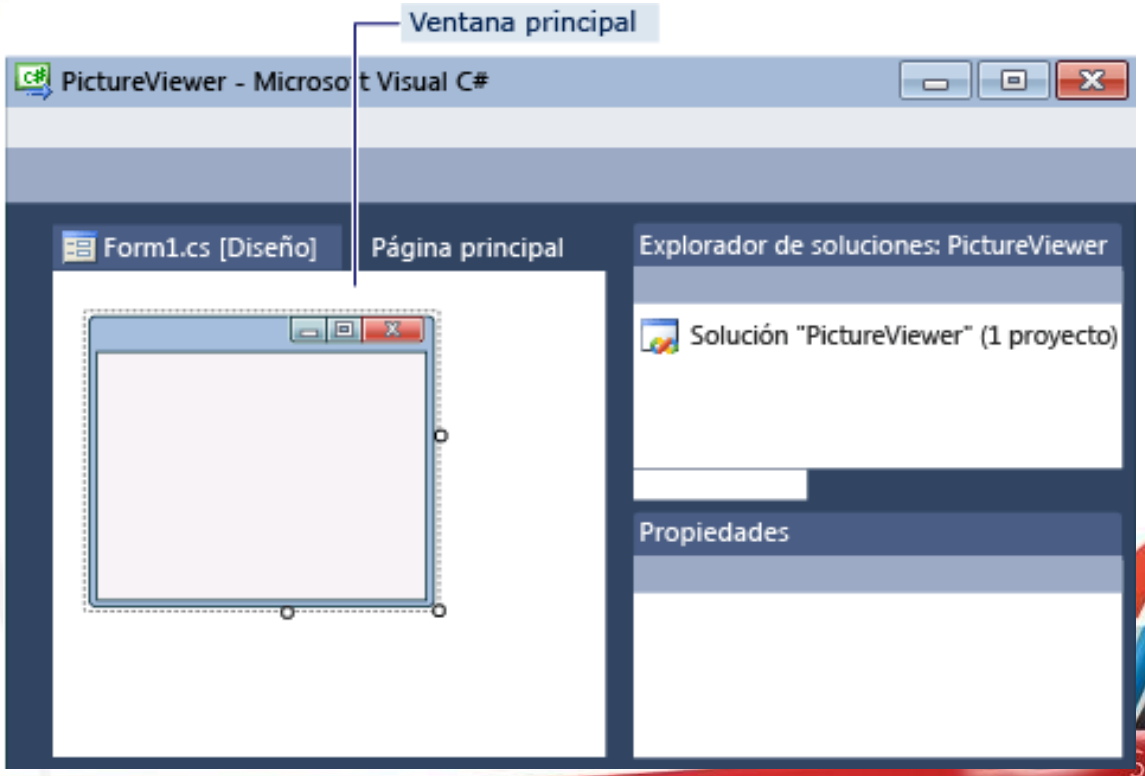
2.En la lista de plantillas, elija el icono **Aplicación de Windows Forms**. Asigne un nombre al nuevo formulario PictureViewer y, a continuación, elija el botón **Aceptar**.

3.Visual Studio crea una solución para el programa. Una solución actúa como un contenedor de todos los proyectos y archivos necesarios para el programa. Estos términos se explicarán con más detalle en secciones posteriores de este tutorial.

4.En la ilustración siguiente se muestra lo que debería aparecer ahora en la interfaz de Visual Studio.

La interfaz contiene tres ventanas: una ventana principal, el **Explorador de soluciones** y la ventana **Propiedades**.

Si falta alguna de estas ventanas, restaure el diseño de ventana predeterminado; para ello, en la barra de menús, elija **Ventana**, **Restablecer diseño de la ventana**. También puede mostrar ventanas mediante comandos de menú. En la barra de menús, elija **Ver**, **Ventana Propiedades**o **Explorador de soluciones**. Si hay otras ventanas abiertas, ciérrelas eligiendo el botón **Cerrar** (x) de la esquina superior derecha.



•**Ventana principal** En esta ventana, realizará la mayor parte del trabajo, como ejecutar formularios y editar código. En la ilustración, la ventana muestra un formulario en el Editor de formularios.

•**Ventana Explorador de soluciones** En esta ventana, puede ver todos los elementos de la solución y navegar por ellos. Al elegir un archivo, cambia el contenido de la ventana **Propiedades**.

•**Ventana Propiedades** En esta ventana, puede cambiar las propiedades de los elementos elegidos en las otras ventanas. Por ejemplo, si elige Form1, puede cambiar el título estableciendo la propiedad **Text** y el color de fondo estableciendo la propiedad **Backcolor**.

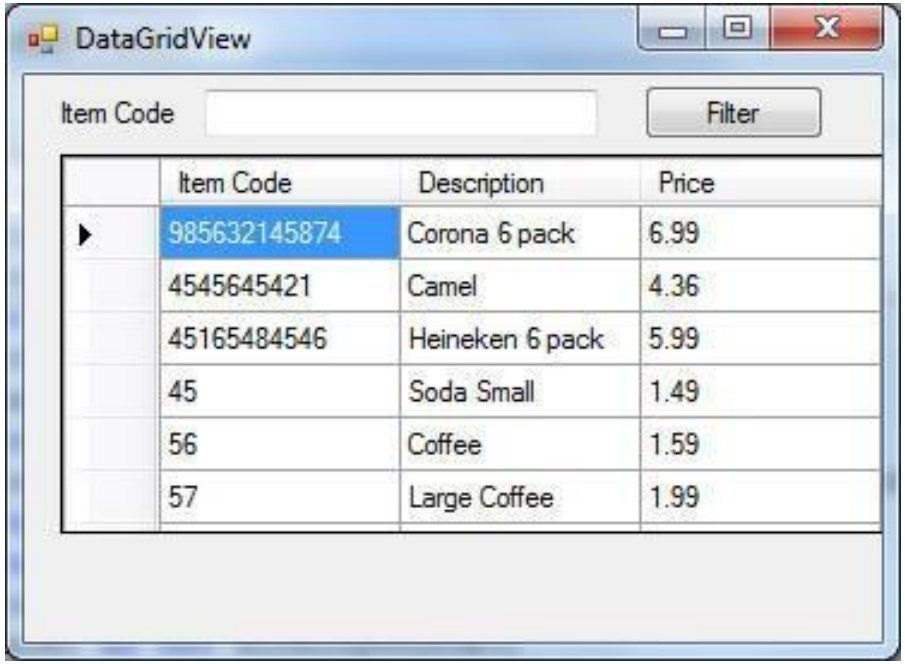
2.5.- Controles Comunes

Los formularios Windows Forms proporcionan controles y componentes que realizan varias funciones. En la tabla siguiente se muestran los controles y componentes de formularios Windows Forms según la función general. Además, donde existen varios controles que atienden la misma función, se muestra el control recomendado con una nota en relación con el control al que reemplaza. En una tabla subsiguiente independiente, se muestran los controles reemplazados con sus reemplazos recomendados.

DataGridView

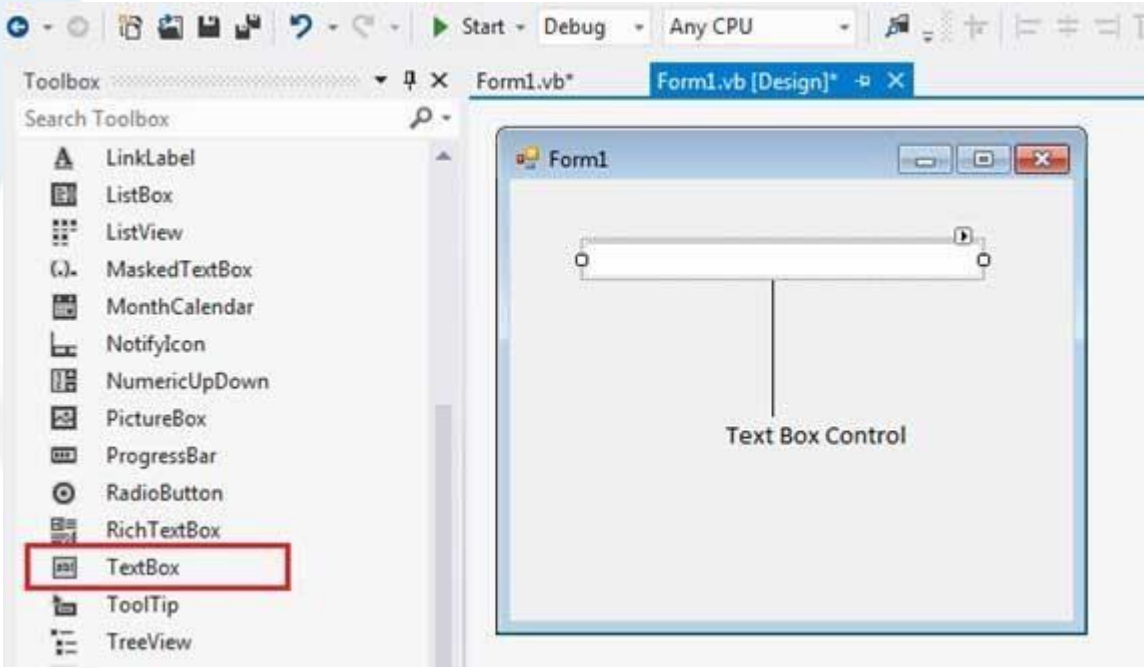
El control DataGridView proporciona una tabla personalizable para mostrar datos.

La clase DataGridView habilita la personalización de celdas, filas, columnas y bordes.



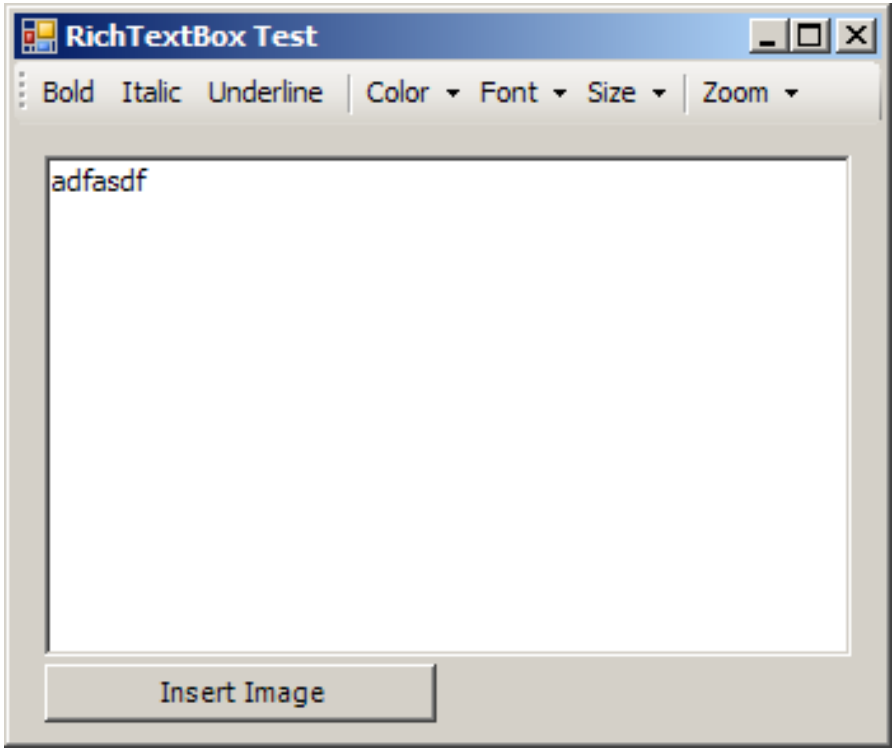
TextBox

Muestra texto escrito en tiempo de diseño que puede ser editado por los usuarios en tiempo de ejecución o ser modificado mediante programación.



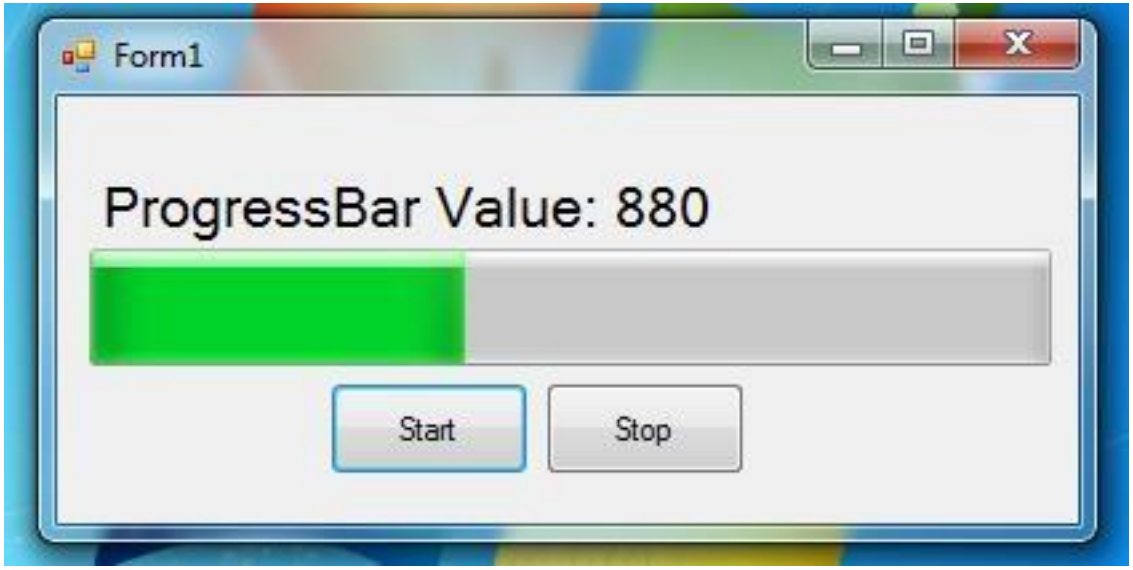
RichTextBox

Habilita la presentación de texto sin formato o de texto enriquecido (RTF).



ProgressBar

Muestra el progreso actual de una operación al usuario



ComboBox

Muestra una lista desplegable de elementos.