RJ CODE
Advance

M1

RJ CODE MODERN UI

Custom themes, styles, forms and controls for WinForm + Source code

PROGRAMMING TUTORIALS

Simple, easy, fast and fun learning

Contenido

1. INTRODUCCIÓN 1
2. CARACTERÍSTICAS 2
3. REQUISITOS 2
4. SOLUCIONAR ERRORES COMUNES 2
   4.1. No se pudo procesar el archivo - Marca de la Web 2
5. LIBRERÍA FONTAWESOME.SHARP 3
   5.1. ¿Qué es FontAwesome? 3
   5.2. ¿Qué es FontAwesome.Sharp? 3
6. UTILIDADES 4
   6.1. Diagrama de clases 4
   6.2. Editor de color - Clase ColorEditor 5
   6.3. Esquinas redondeadas - Clase RoundedCorner 5
7. CONFIGURACIÓN DE APARIENCIA 6
   7.1. Diagrama de clases 6
   7.2. Lista de colores - Clase RJColors 8
   7.3. Archivo de configuración – UIAppearanceSettings 10
   7.4. Temas - Enumeración UITheme 10
   7.5. Estilos - Enumeración UIStyle 10
   7.6. Apariencia de Interfaz de Usuario - Estructura UIAppearance 11
   7.7. Administrador de configuración - Clase SettingsManager 13
   7.8. Formulario de configuración - Clase RJFormSettings 13
   7.9. Clase Program 14
   7.10. Conclusiones 14
8. CONTROLES PERSONALIZADOS 15
   8.1. Estilos de control – Enumeración ControlStyle 17
   8.2. Controles extendidos 17
      8.2.1. Controles comunes 17
         8.2.1.1. Diagrama de clases 17
         8.2.1.2. Clase RJ Button 19
         8.2.1.3. Clase RJ CheckBox 20
         8.2.1.4. Clase RJ Label 21
         8.2.1.5. Clase RJ RadioButton 22
      8.2.2. Controles contenedores 22
         8.2.2.1. Diagrama de clases 23
         8.2.2.2. Clase RJ ImageColorOverlay 24

8.2.2.3. Clase RJ Panel 24
8.2.3. Controles de datos 25
8.2.3.1. Diagrama de clases 25
8.2.3.2. Clase RJ DataGridView 26
8.2.3.3. Clase RJ Chart 27
8.2.4. Controles de menú 28
8.2.4.1. Diagrama de clases 28
8.2.4.2. Clase RJ DropdownMenu 30
8.2.4.3. Clase RJ MenuButton 32
8.2.4.4. Clase RJ MenuIcon 33
8.2.5. Controles especiales 34
8.2.5.1. Diagrama de clases 34
8.2.5.2. Clase RJ CircularPictureBox 36
8.2.5.3. Clase RJ ToggleButton 36
8.2.5.4. Clase RJ TrackBar 37
8.3. Controles compuestos (User Control) 38
8.3.1. Diagrama de clases 39
8.3.2. Clase RJ ComboBox 41
8.3.3. Clase RJ DatePicker 42
8.3.4. Clase RJ TextBox 44
8.4. Componentes 46
8.4.1. Diagrama de clases 47
8.4.2. Clase RJ DragControl 47
8.5. ¿Cómo usarlos? 48
9. FORMULARIOS PERSONALIZADOS 49
9.1. Diagrama de clases (Contraído) 49
9.2. Formularios personalizados Base 50
9.2.1. Diagrama de clases (Expandido) 50
9.2.2. Formulario base - Clase RJBaseForm 51
9.2.3. Formulario principal - Clase RJMainForm 52
9.2.4. Formulario secundario - Clase RJChildForm 54
9.2.5. ¿Cómo usarlos? 55
9.3. Formularios personalizados Derivados 56
9.3.1. Diagrama de clases (Expandido) 56
9.3.2. Formulario de inicio de sesión - Clase LoginForm 57
9.3.3. Formulario principal - Clase MainForm 58

9.3.4. Formulario de impresión - Clase RJPrintForm 58
9.3.5. Formulario de configuración - Clase RJSettingsForm 59
10. CUADRO DE MENSAJE PERSONALIZADO 60
10.1. Diagrama de clases 60
10.2. Clase RJMessageForm 62
10.3. Clase RJMessageBox 63

1. INTRODUCCIÓN

Hola bienvenido(a) a este **tutorial escrito del proyecto RJ Code Modern UI**, plantilla con diseño **moderno y plano**. Para el diseño se crean **formularios personalizados, controles personalizados** y los componentes necesarios para la **configuración de apariencia de la aplicación**.

Este proyecto incluye:

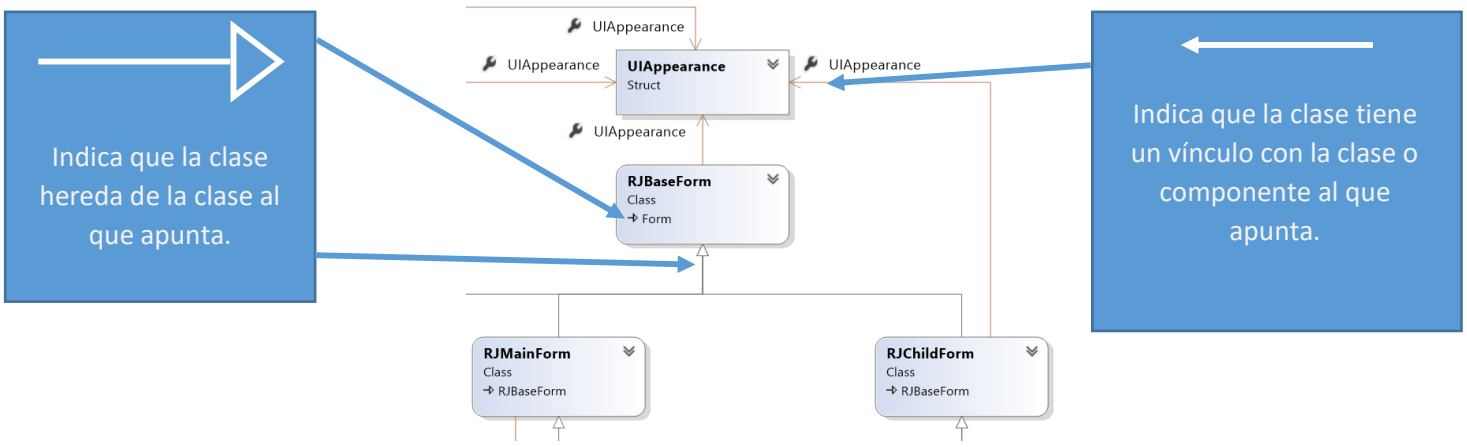
- ✓ Incluye **3 Formularios Base** Personalizados.
- ✓ Incluye **18 Controles Personalizados**.
- ✓ Incluye **2 Utilidades**
- ✓ Incluye **7** Componentes para la **Configuración de Apariencia** (Incluido un formulario prefabricado para configuración de la aplicación)
- ✓ Incluye **8 Formularios prefabricados** (Además, más **7 formularios** de prueba para los formularios y controles personalizados con toda la demostración de diseños y estilos).
- ✓ Y se incluye esta **documentación completa** que incluye también los diagramas de clases para el fácil entendimiento.

Este es un **proyecto basado en la programación orientada a objetos** (POO), por lo que se utiliza constantemente el pilar de **la herencia, encapsulamiento y polimorfismo**, además de incluir los diagramas de clases, ya que es el componente principal del modelado orientado a objetos.

Antes de comenzar a leer la documentación y mirar el código fuente del proyecto quiero que tengas muy en cuenta lo siguiente.

Relaciones – Diagrama de clases

Es muy importante conocer el **significado de las relaciones en un diagrama de clases**, ello te ayuda a comprender y saber qué tipo de relación tiene las clases o componentes.



Palabra clave this

Personalmente me gusta usar la palabra clave **this**, para indicar que el campo o propiedad al que accedo se encuentra en una clase base (Superclase o Padre), en el proyecto lo utilizo a menudo, por lo tanto **ten en cuenta que el campo o propiedad se encuentra en una clase base**.

2. CARACTERÍSTICAS

- IDE: Visual Studio 2012
- Marco: .NET Framework 4.5
- Lenguaje: C# 5.0
- Plataforma: Windows Form
- Paquete Nuget 1: FontAwesome.Sharp 5.15.3

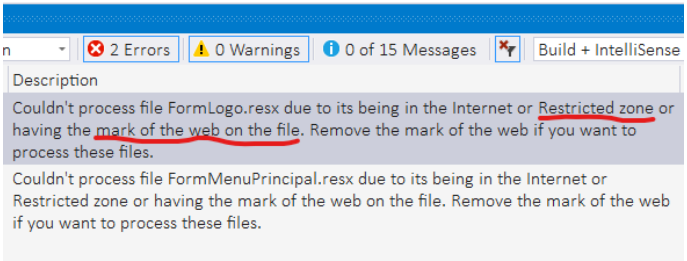
3. REQUISITOS

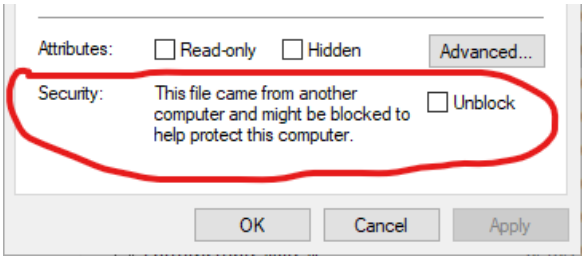
- ✓ Visual Studio 2012 o Superior (No recomiendo VS 2015 Update 1)
- ✓ .NET Framework 4.5 o Superior

4. SOLUCIONAR ERRORES COMUNES

4.1. No se pudo procesar el archivo - Marca de la Web

A menudo me envían un mensaje al correo sobre el siguiente error.





***"ERROR**: No se pudo procesar el archivo "CualquierArchivo.resx" debido a que está en **Internet o en una zona restringida** o porque tiene la **marca de la web** en el archivo. **Elimine la marca de la web si desea procesar estos archivos**."*

Este es un error muy frecuente (**Identificador de zona o marca de la web**), **sucede cuando se descarga un proyecto de Visual Studio o cualquier archivo desde internet**, estos archivos se marcan con el identificador de zona mediante flujos de datos alternativos.

Solución:

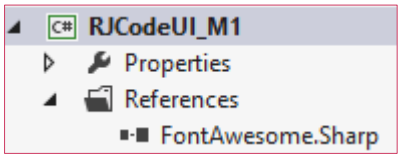
A. **Opción 1:** Desbloquear el archivo **desde las propiedades** (Ver imagen de arriba-derecha), recomiendo hacer ello en el archivo comprimido (.rar, .zip) antes de descomprimir cualquier proyecto, de lo contrario tendrás que desbloquear cada archivo del proyecto.

B. **Opción 2:** Desbloquear todos los archivos de todo el proyecto **mediante PowerShell**, ejecuta el siguiente **comando**: *Get-ChildItem -Recurse -Path 'Ruta de la carpeta del proyecto' |Unblock-File*

C. **Opción 3:** Descarga y usa la herramienta **RJ ZoneID Remover** que creé especialmente para estos casos, desbloquea cualquier archivo y/o carpetas masivamente.

5. LIBRERÍA FONTAWESOME.SHARP

Este proyecto usa la librería FontAwesome.Sharp para agregar icono a los formularios base (RJMainForm - RJChildForm) y controles de usuario (RJComboBox - RJDatePicker), y extender el control IconButton e IconPictureBox.



Esta librería fue creada por mkoertgen, tiene actualizaciones constantes y espero que siga así por un largo tiempo, puedes descargarlo desde GitHub o Nuget para incluirlo en tu proyecto (En este caso no es necesario, la librería ya está descargada e integrada en el proyecto).

Sobre el autor:

- Nombre: Marcel Körtgen
- Perfil GitHub: https://github.com/mkoertgen
- Perfil Nuget: https://www.nuget.org/profiles/mkoertgen
- Cantidad de proyectos en Nuget: 18

Descargar librería FontAwesome.Sharp:

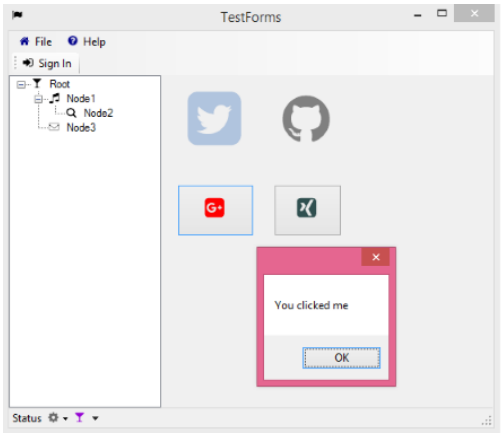
- GitHub: https://github.com/awesome-inc/FontAwesome.Sharp
- Nuget: https://www.nuget.org/packages/FontAwesome.Sharp/

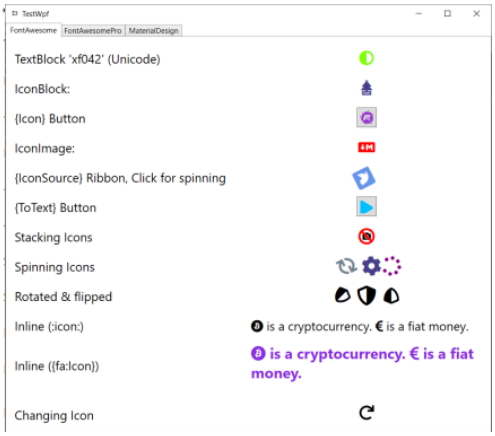
5.1. ¿Qué es FontAwesome?

FontAwesome es una herramienta que permite **crear iconos basados en vectores a CSS a partir de un formato de fuentes de texto**, donde es realmente sencillo colocar iconos con un color y tamaño personalizado, estoy seguro que muchos de ustedes lo han usado al crear páginas web o aplicaciones web o al menos lo han oído. Actualmente, FontAwesome tiene **1609** iconos en su versión GRATUITA y **6256** iconos en su versión PRO. Bueno existen muchas más herramientas similares como; Material Icons, Captain Icons, Octicons, Typicons, Zondicons, Devicons, etc. Sin embargo, ninguno de estas herramientas tiene soporte para aplicaciones de escritorio, **solamente se pueden incluir en páginas web** y hojas de estilo.

5.2. ¿Qué es FontAwesome.Sharp?

Como se dijo anteriormente, la herramienta *FontAwesome* solamente puede ser usado en páginas web o aplicaciones web. Por lo tanto, **FontAwesome.Sharp** es una **librería de controles** que te permite incrustar iconos de *FontAwesome* en tus **aplicaciones de escritorio WPF o Windows Forms,** además si tienes una licencia de FontAwesome PRO, puedes usar los iconos en esta librería sin ningún problema.



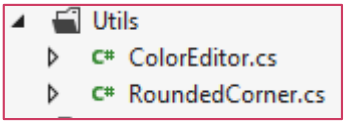


Actualmente cuenta con 2 controles que puedes ser integrado a la caja de herramientas y 4 Componentes que puedes usar mediante código.

- IconButton (Control)
- IconPictureBox (Control)
- IconSplitButton
- IconToolStripButton
- IconDropDownButton
- IconMenuItem

Para **obtener más información y ejemplos** puedes visitar el **repositorio de Marcel Körtgen** en GitHub, también puede ver mis pequeños **video tutoriales** de como descargar, instalar y usar FontAwesome.Sharp en C# o Visual Basic con Windows Form.

6. UTILIDADES



Generalmente las utilidades en un proyecto son **clases estáticas que definen funciones/métodos** estáticos que pueden ser usados de manera repetitiva en todo el proyecto o en un ensamblado específico para **realizar operaciones sin necesidad de instanciar**, ello nos ahorra mucho tiempo y evita crear objetos innecesarios que puede ser contraproducente con el rendimiento, como siempre he mencionado, no es recomendable crear instancias por doquier.

Para crear estos clases/métodos, puedes colocar el **nombre que prefieras** a la carpeta (en mi caso Utils), no hay ninguna regla que indique como nombrarlo. Bueno, en este caso creé 2 clases de utilidades; la **clase ColorEditor** y la **clase RoundedCorner**.

6.1. Diagrama de clases

Diagrama de clases contraído

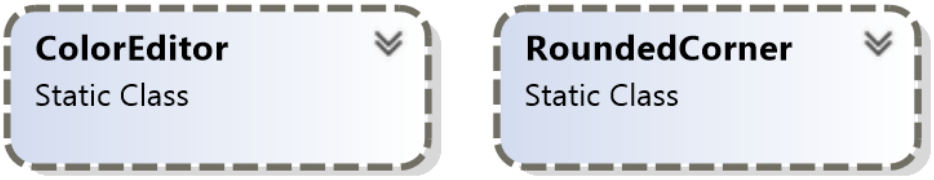
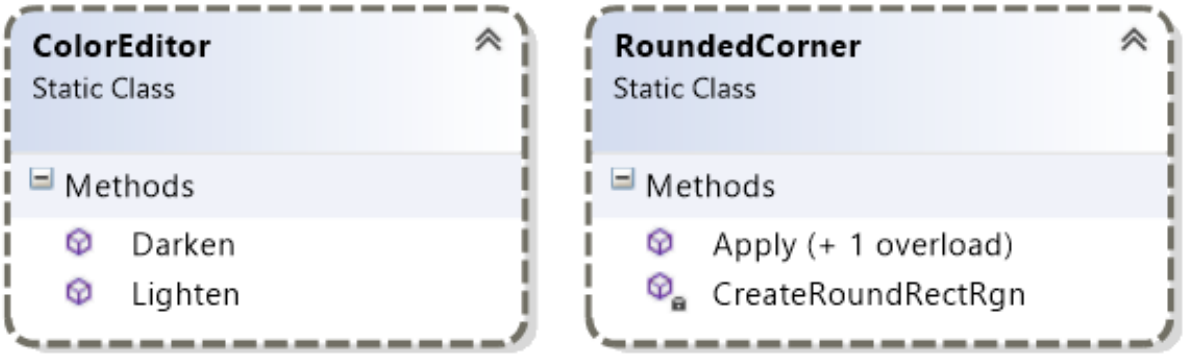


Diagrama de clases expandido



6.2. Editor de color - Clase ColorEditor

La clase Editor de Color (*ColorEditor.cs*) **permite aclarar u oscurecer un color** especificado con valor de intensidad especificado, está basado en el ejemplo de Pavel Vladov.

Métodos

`Darken(Color color, ushort percentage)`	Permite oscurecer un color en específico con un valor de oscurecimiento en específico, el valor máximo es 100%.
`Lighten(Color color, ushort percentage)`	Permite aclarar un color en específico con un valor de aclarado en específico, el valor máximo es 100%.

Ambos métodos retornan un color con los cambios realizados.

¿Cómo usarlo?

Usar estos métodos es realmente sencillo, simplemente llama la clase estática ColorEditor, invoca uno de sus métodos y envía un color y valor de intensidad que deseas, por ejemplo:

```
//Oscurecer color 10%
this.ForeColor = ColorEditor.Darken(textColor, 10);
//Aclarar color 21%
MyButton.BackColor= ColorEditor.Lighten(UIAppearance.StyleColor, 21);
```

6.3. Esquinas redondeadas - Clase RoundedCorner

La clase Esquinas redondeados (*RoundedCorner.cs*) permite crear controles o formularios con esquinas redondeadas.

Métodos

`CreateRoundRectRgn(int nLeftRect, int nTopRect, int nRightRect, int nBottomRect, int nWidthEllipse, int nHeightEllipse)`	Método externo que permite crear un rectángulo con esquinas redondeados con un valor de alto y ancho de elipse especificados.
`Apply(Control control, int radius)`	Permite crear la región de un control con esquinas redondeados con un radio especificado.
`Apply(Form form, int radius)`	Permite crear la región de un formulario con esquinas redondeados con un radio especificado.

¿Cómo usarlo?

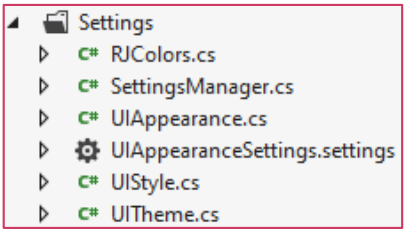
Llama la clase RoundedCorner, invoca el método Apply, y envía una instancia de formulario o control y un radio como parámetros, por ejemplo:

```
//Aplicar esquinas redondeadas con un radio de 15 a un control
RoundedCorner.Apply(MyControl, 15);
//Aplicar esquinas redondeadas con un radio de 8 a un formulario
RoundedCorner.Apply(MyForm, 8);
```

7. CONFIGURACIÓN DE APARIENCIA

A la gran mayoría nos gusta personalizar la apariencia de nuestro sistema operativo Windows, Android, Apple o una aplicación en específico, cambiando el color de ventanas, fondo, iconos, barra de menú, barra de títulos, etc. donde sea de nuestro agrado y nos sintamos más cómodo.

Por ello, este proyecto prioriza e **integra la función de configurar la apariencia de la interfaz de usuario,** permitiendo **cambiar el tema, estilo de color, diseños, ancho de bordes, color de bordes**, y entre otros, aplicando los ajustes de personalización a **formularios y controles**.



7.1. Diagrama de clases

Diagrama de clases contraído

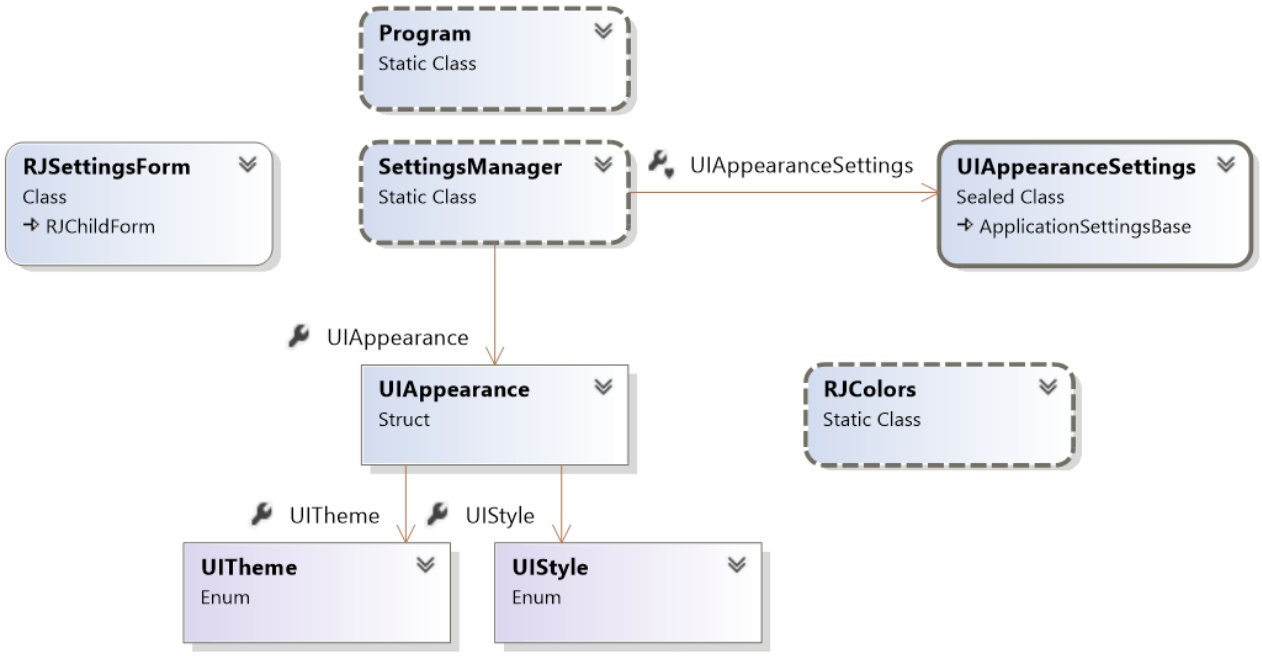
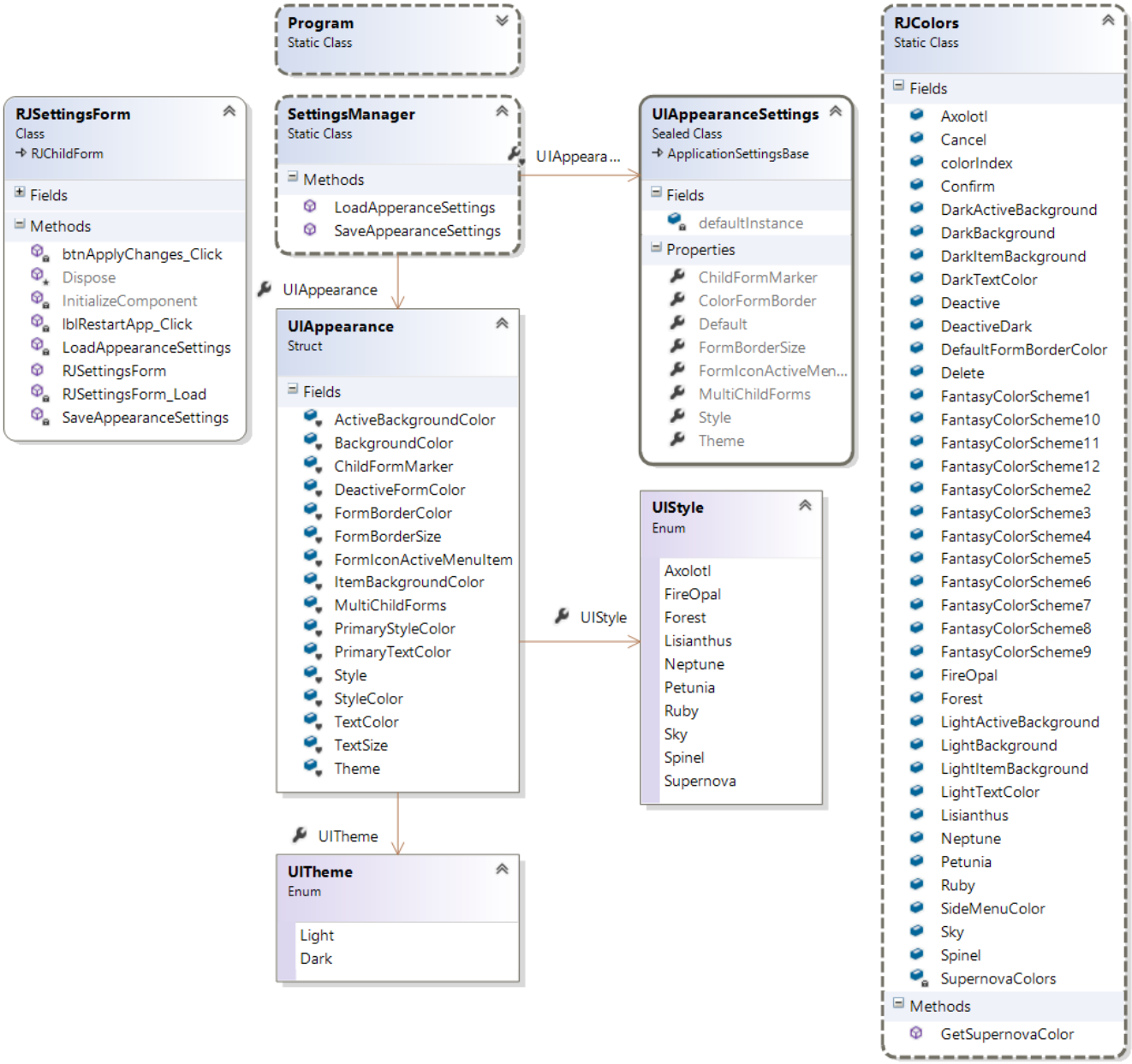


Diagrama de clases expandido



7.2. Lista de colores - Clase RJColors

La clase estática RJColors simplemente define una lista de colores para el color de tema, color de estilos, color de texto, entre otros, e implementa un método para obtener un color para el estilo supernova.

Campos

`colorIndex`	Obtiene o establece el índice de color de la lista de colores del estilo supernova.
`SideMenuColor`	Obtiene o establece el color del menú lateral.
`DefaultFormBorderColor`	Establece el color de borde por defecto de un formulario.
`Axolotl`	Obtiene o establece el color de los estilos.
`FireOpal`	
`Forest`	
`Lisianthus`	
`Neptune`	
`Petunia`	
`Ruby`	
`Sky`	
`Spinel`	
`FantasyColorScheme1`	Obtiene o establece el esquema de color fantasía.
`FantasyColorScheme2`	
`FantasyColorScheme3`	
`FantasyColorScheme4`	
`FantasyColorScheme5`	
`FantasyColorScheme6`	
`FantasyColorScheme7`	
`FantasyColorScheme8`	
`FantasyColorScheme9`	
`FantasyColorScheme10`	

FantasyColorScheme11	
FantasyColorScheme12	
DarkBackground	Obtiene o establece el color para el tema oscuro.
DarkItemBackground	
DarkActiveBackground	
DarkTextColor	
LightBackground	Obtiene o establece el color para el tema claro.
LightItemBackground	
LightActiveBackground	
LightTextColor	
Delete	Obtiene o establece el color de acciones
Confirm	
Cancel	
Deactive	
DeactiveDark	

Métodos

GetSupernovaColor()	Permite obtener un color de la lista de colores del estilo supernova.

¿Cómo usar?

Puedes obtener un color y establecer a cualquier propiedad de color del control, por ejemplo:

```
MyButton.BackColor= RJColors.GetSupernovaColor();
```

7.3. Archivo de configuración – UIAppearanceSettings

El archivo de configuración de los ajustes de apariencia (*UIAppearanceSettings.settings*) permite almacenar y recuperar la configuración de las propiedades de apariencia de los usuarios.

Propiedades

`ChildFormMarker`	Obtiene o establece si el marcador de formulario secundario se mostrará en el botón de menú del menú lateral del formulario principal.
`ColorFormBorder`	Obtiene o establece si el borde de los formularios tendrá color, caso contrario se mostrará con un color por defecto.
`Default`	Obtiene o establece el valor de las propiedades del archivo de configuración.
`FormBorderSize`	Obtiene o establece el ancho de borde de los formularios.
`FormIconActiveMenuItem`	Obtiene o establece si el icono del formulario asociado se mostrará en el elemento de menú (`ToolStripMenuItem`) activo.
`MultiChildForms`	Obtiene o establece si el usuario pueda abrir varios formularios secundarios dentro del panel escritorio del formulario principal, caso contrario, solamente se podrá abrir un único formulario.
`Style`	Obtiene o establece el estilo de la interfaz de usuario.
`Theme`	Obtiene o establece el tema de la interfaz de usuario.

7.4. Temas - Enumeración UITheme

La enumeración de Temas (*UITheme.cs*) define temas para la aplicación.

Campos

`Dark`	El color de fondo de los formularios y elementos tienen un aspecto de color oscuro.
`Light`	El color de fondo de los formularios y elementos tienen un aspecto de color claro.

7.5. Estilos - Enumeración UIStyle

La enumeración de Estilos (*UIStyle.cs*) define estilos para la aplicación, un estilo define un color de acento o apariencia para los elementos de la aplicación, por ejemplo, la barra de título y borde de los formularios, el fondo, borde, icono y otros elementos de los controles.

Campos

`Axolotl`	La barra de título del formulario y la apariencia de los controles tienen un color de tono rosa opaco.
`FireOpal`	La barra de título del formulario y la apariencia de los controles tienen un color de tono naranja.
`Forest`	La barra de título del formulario la apariencia de los controles tienen un color de tono verde.
`Lisianthus`	La barra de título del formulario y la apariencia de los controles tienen un color de tono purpura.
`Neptune`	La barra de título del formulario y la apariencia de los controles tienen un color de tono azul.
`Petunia`	La barra de título del formulario y la apariencia de los controles tienen un color de tono morado.
`Ruby`	La barra de título del formulario y la apariencia de los controles tienen un color de tono rojo.
`Sky`	La barra de título del formulario y la apariencia de los controles tienen un color de tono celeste.
`Spinel`	La barra de título del formulario y la apariencia de los controles tienen un color de tono rosa claro.
`Supernova`	La barra de título del formulario tiene un color similar al tema, y la apariencia de algunos controles principales pueden tener cualquier color de la lista de colores del estilo supernova.

Para obtener la vista previa del color exacto de cada estilo, ver punto 7.2– Lista de colores

7.6. Apariencia de Interfaz de Usuario - Estructura UIAppearance

La estructura Apariencia de Interfaz de Usuario (*UIAppearace.cs*) **almacena valores de la configuración de apariencia y otros valores de apariencia** (*es decir, que no se almacenan en el archivo de configuración, pero se establecen a partir de ello, tales como: color de texto primario, color de texto normal, color de estilo primario, color de estilo normal, color de fondo, entre otros, para más información ver siguiente punto (7.7)*) para la apariencia de los formularios y controles en tiempo de ejecución.

Campos

`ActiveBackgroundColor`	Obtiene o establece el color de fondo de la aplicación en su estado activo o resaltado.
`BackgroundColor`	Obtiene o establece el color de fondo de la aplicación.

DeactiveFormColor	Obtiene o establece el color de la barra de título del formulario cuando está en estado desactivo o pierde el foco.
ChildFormMarker	Obtiene o establece si el marcador de formulario secundario se mostrará en el botón de menú del menú lateral del formulario principal.
FormBorderColor	Obtiene o establece el color del borde de los formularios.
FormBorderSize	Obtiene o establece el ancho de borde de los formularios.
FormIconActiveMenuItem	La barra de título del formulario y la apariencia de los controles tienen un color de tono rojo.
ItemBackgroundColor	La barra de título del formulario y la apariencia de los controles tienen un color de tono celeste.
MultiChildForms	Obtiene o establece si el usuario pueda abrir varios formularios secundarios dentro del panel escritorio del formulario principal, caso contrario, solamente se podrá abrir un único formulario.
PrimaryStyleColor	La barra de título del formulario tiene un color similar al tema, y la apariencia de algunos controles principales pueden tener cualquier color de la lista de colores del estilo supernova.
PrimaryTextColor	
Style	Obtiene o establece el estilo de la interfaz de usuario de la aplicación.
StyleColor	Obtiene o establece el color de estilo para los elementos de la aplicación (Barras títulos y apariencia de controles)
TextColor	Obtiene o establece el color de texto de la interfaz de usuario, principalmente aplica a texto tipo párrafo por ejemplo: Labels, RadioButton, TextBoxs.
TextSize	Obtiene o establece el tamaño de texto, aplica a la mayoría de los controles personalizados al ser agregados al formulario por primera vez, luego puede ser cambiado desde las propiedades, **excepto el control RJLabel con estilo Normal.**
Theme	Obtiene o establece el tema de la interfaz de usuario de la aplicación.

7.7. Administrador de configuración - Clase SettingsManager

La clase administrador de configuración se encarga de guardar la configuración de apariencia en el archivo de configuración y establecer todos los valores de configuración en la estructura apariencia de interfaz de usuario.

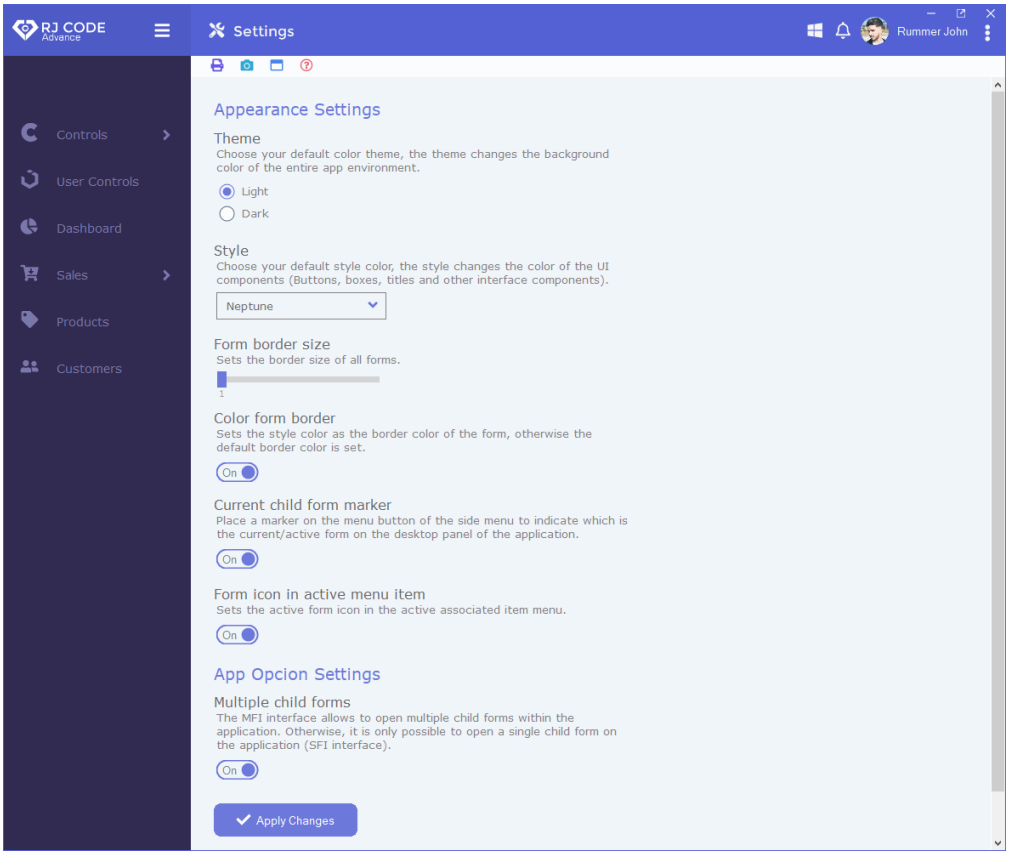
Métodos

`SaveAppearanceSettings(`
`int theme,`
`int style,`
`int formBorderSize,`
`bool colorFormBorder,`
`bool childFormMarker,`
`bool formIconActiveMenuItem,`
`bool multiChildForms)`
	Guarda los datos de configuración de apariencia en el archivo de configuración (*UIAppearanceSettings.settings*) permanentemente.

`LoadApperanceSettings()`	Se encarga de **obtener los datos de la configuración** de apariencia del archivo de configuración, y **establecer los valores de la estructura Apariencia de Interfaz de Usuario**, tales como: establecer el color de estilo primario y normal, establecer o no el color de borde del formulario, establecer el color de texto primario y normal, y entro otros, todo ello de acuerdo al tema establecido.

7.8. Formulario de configuración - Clase RJFormSettings

El formulario de configuración se encarga de visualizar la configuración de actual, hacer cambios y guardar la configuración modificada en el archivo de configuración (*UIAppearanceSettings.settings*).



Métodos

`LoadApperanceSettings()`	Se encarga de **obtener los datos de la configuración** actual de la estructura Apariencia de Interfaz de Usuario (*UIAppearance.cs*) y visualizarla en la interfaz del formulario, *ver imagen anterior*.
`SaveAppearanceSettings()`	**Guarda los cambios realizados** en el archivo de configuración (*UIAppearanceSettings.settings*) a través del administrador de configuración.
`RJSettingsForm_Load(` `object sender, EventArgs e)`	Invoca el método LoadApperanceSettings() para cargar la configuración de apariencia actual.
`btnApplyChanges_Click(` `object sender, EventArgs e)`	Invoca el método SaveAppearanceSettings() para guardar los cambios.
`lblRestartApp_Click(` `object sender, EventArgs e)`	Reinicia la aplicación para visualizar los cambios.

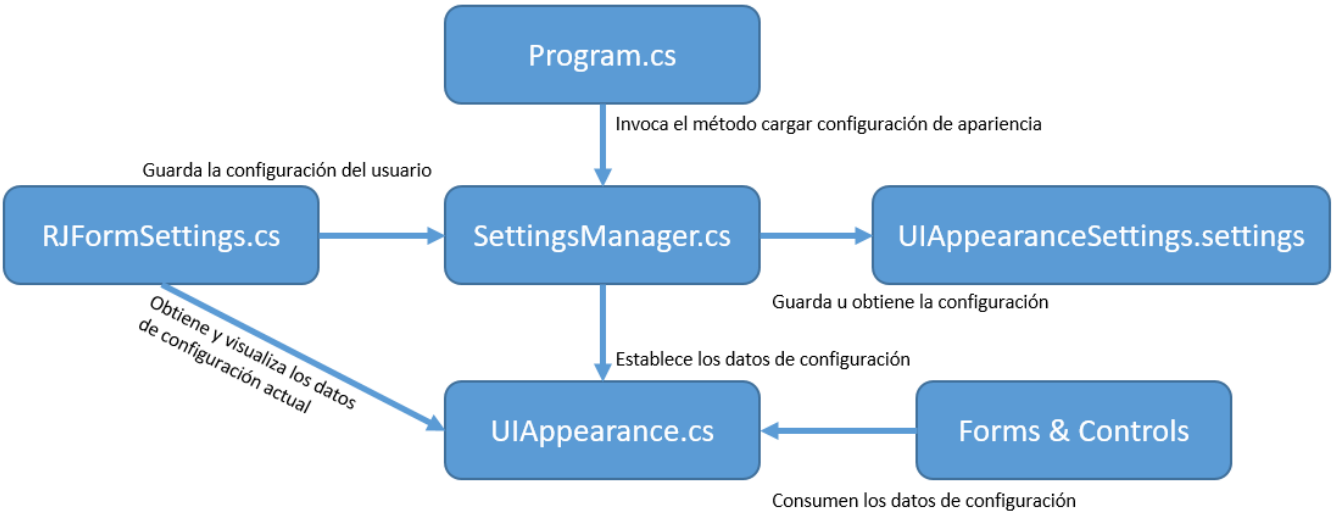
7.9. Clase Program

La clase Program simplemente se encarga de invocar el método LoadAppearanceSettings() de la clase administrador de configuración para cargar los datos del archivo de configuración en la estructura apariencia de la interfaz de usuario antes de mostrar un formulario.

```
static void Main()
{
            Settings.SettingsManager.LoadApperanceSettings();
```

7.10. Conclusiones

El funcionamiento de la configuración de apariencia es simple:



8. CONTROLES PERSONALIZADOS

Se pueden crear controles personalizados extendiendo (Herencia) un control existente de .NET Framework. Se puede hacer mediante 3 maneras: Extendidos, personalizados o compuestos.

Controles extendidos

Un control extendido es simplemente cuando se hereda cualquier control de Windows Forms existente. Este método permite conservar toda la funcionalidad del control, para luego ampliarla agregando propiedades, métodos y opcionalmente anular el evento Paint para ampliar o volver a dibujar completamente la apariencia del control.

```
public class MyCustomButton : Button
```

Controles personalizados

Puede crear un control desde cero heredando la clase Control, esta clase proporciona toda la funcionalidad básica para los controles, sin embargo no incluye la interfaz gráfica, por lo que es necesario dibujar el control y agregar las funciones/propiedades necesarios, ello requiere mucho esfuerzo.

```
public class MyZoomControl : Control
```

Controles compuestos o Control de Usuario

Un control compuesto se crea mediante múltiples controles existentes de Windows Form, para ello puedes agregar un UserControl desde el diseñador y agregar controles, o puedes heredar la clase UserControl y agregar controles mediante código, los controles constituyentes del control de usuario conservan toda su funcionalidad donde puedes exponer y enlazar propiedades, además de asociar o crear los eventos necesarios.

La clase UserControl es básicamente un contenedor para otros controles, recomiendo usar este método cuando es realmente necesario y como última opción, ya que es más pesado que un control extendido o personalizado.

```
public class MyImageSlider : UserControl
```

Para obtener más información y recomendaciones puedes hacer clic aquí.

Bueno, en este proyecto creé un total de **18 controles personalizados**, **3 de ellos son controles de usuario** (creado por varios controles existentes), **14 de ellos son controles extendidos** (Hereda de un solo control existente), para este caso de la documentación están divididos en **5 categorías** para no enredarse con tantas clases en los diagramas, poder visualizarla y entender mejor, finalmente **1 Componente** (Los componentes no son controles, ya que **no que tienen una interfaz gráfica** y no es un elemento secundario del formulario, pero puede ser tratado como uno ya se integran a la caja de herramientas de visual estudio y pueden ser arrastrados a los formularios).

NOTA:

***El color de apariencia de los controles es establecido por la configuración de apariencia de la aplicación**, sin embargo, el color de apariencia puede ser personalizado estableciendo la propiedad **Customizable** en verdadero (Customizable= true), o la propiedad **Design** en personalizable (Desing= Customizable) según sea el caso.*

A) Controles extendidos

-Controles comunes

1. RJ Button
2. RJ CheckBox
3. RJ Label
4. RJ RadioButton

-Controles contenedores

5. RJ ImageColorOverlay
6. RJ Panel

-Controles de datos

7. RJ DataDridView
8. RJ Chart

-Controles de menú

9. RJ DropdownMenu
10. RJ MenuButton
11. RJ MenuIcon

-Controles especiales

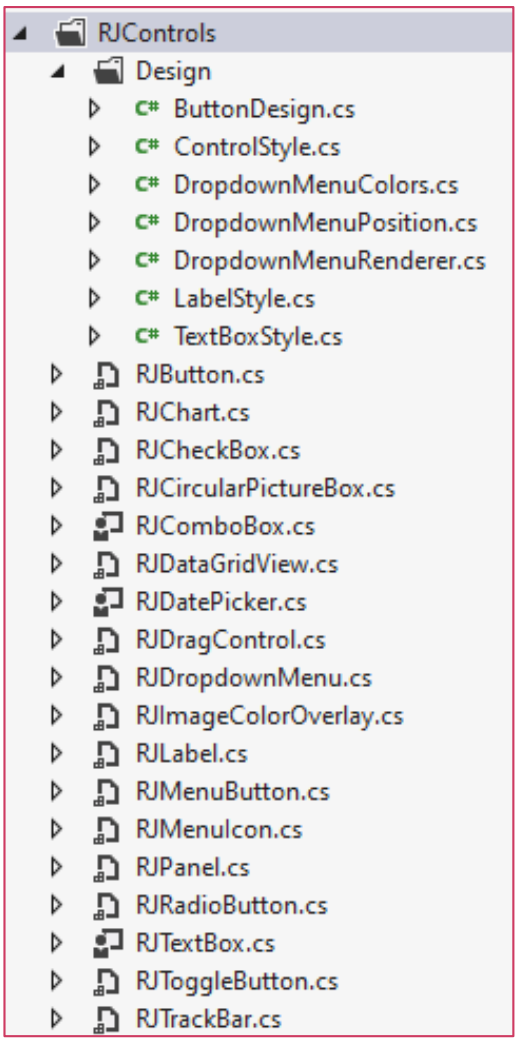
12. RJ ToogleButton
13. RJ TrackBar
14. RJ CircularPictureBox

B) Controles compuestos o Controles de usuario

15. RJ Combo Box
16. RJ Date Picker
17. RJ Text Box

C) Componentes

18. RJ DragControl



8.1. Estilos de control – Enumeración ControlStyle

La enumeración estilos de control (*ControlStyle.cs*) **define estilos de apariencia** para la mayoría de los controles personalizados del proyecto, por ejemplo: botones, cajas de combo, selectores de fecha, casillas de verificación y entre otros.

`Glass`	El color de fondo del control es **transparente** y con borde de color, en este estilo puede cambiar el tamaño de borde.
`Solid`	El color de fondo del control es un color sólido y sin borde, este estilo permite aplicar esquinas redondeados al control, para ello utiliza la clase RoundedCorner de las utilidades.

8.2. Controles extendidos

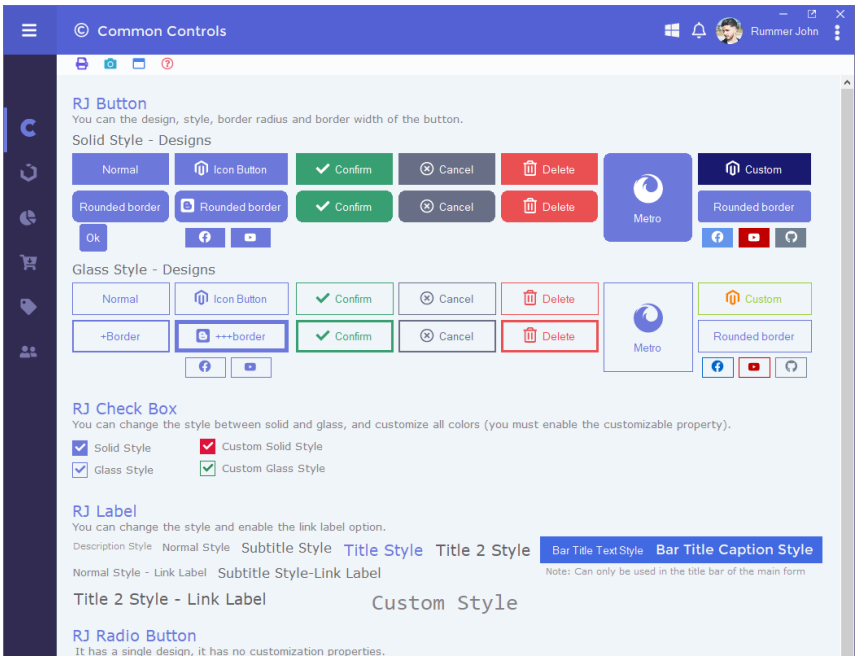
Como dije anteriormente, un control extendido es cuando **hereda de cualquier otro control de Windows Forms** existente, **conserva toda la funcionalidad** del control, para luego ampliarla agregando propiedades, métodos, además de **poder anular el evento Paint** para ampliar o volver a dibujar completamente la apariencia del control.

El color de apariencia de los controles es establecido por la configuración de apariencia de la aplicación, sin embargo, el color de apariencia puede ser personalizado estableciendo la propiedad **Customizable**.

En este proyecto hay **14 controles extendidos,** dividido en **5 categorías:**

8.2.1. Controles comunes

Son controles comunes y los más usados en cualquier formulario, por ejemplo: **TextBox, Label, RadioButton y CheckBox.** A continuación se muestra una **captura de pantalla** de estos controles con todos los diseños y estilos disponibles.



8.2.1.1. Diagrama de clases

Diagrama de clases contraído

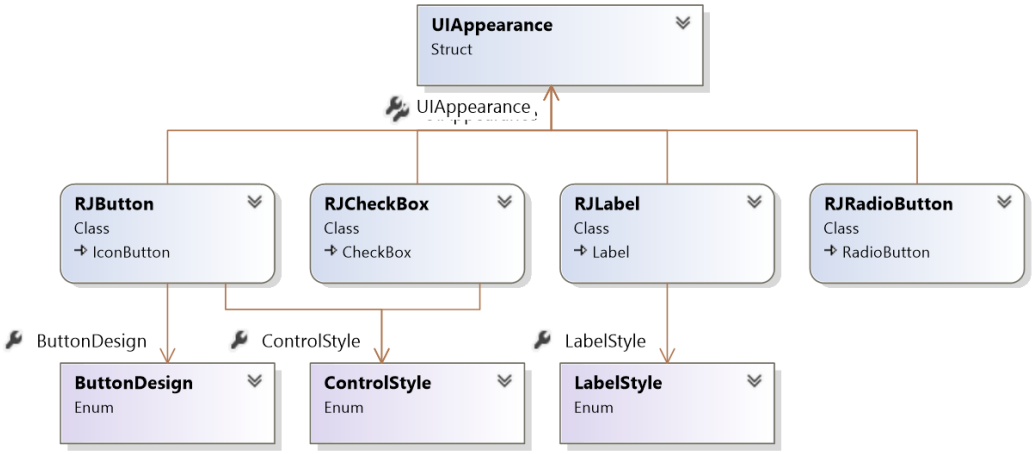
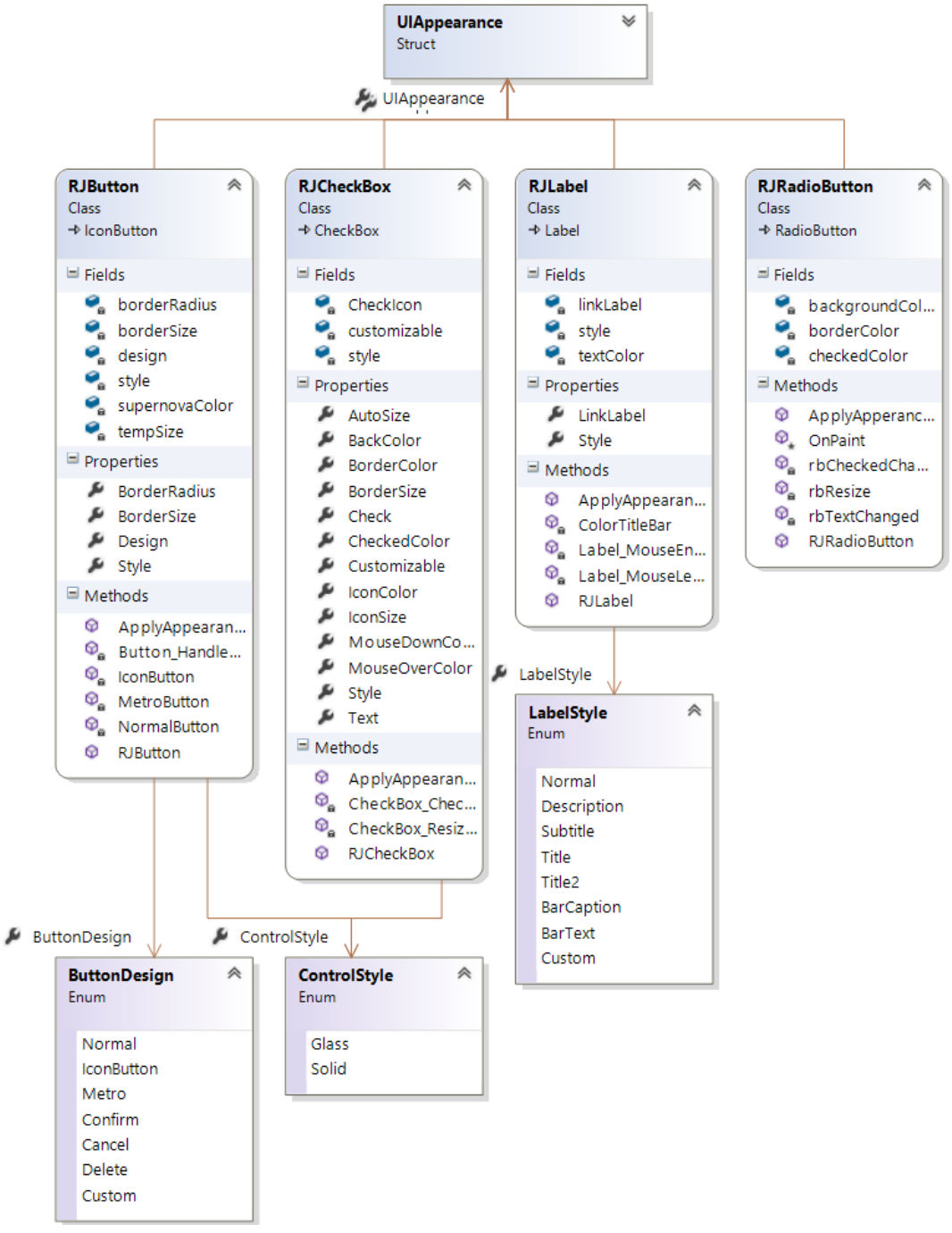


Diagrama de clases expandido



8.2.1.2. Clase RJ Button

Esta clase **hereda** de la clase **IconButton** de la librería **FontAwesome.Sharp**, a la misma, IconButton **hereda** de la clase **Button** de la librería **Windows.Forms**.

El control RJButton implementa **4 propiedades principales de apariencia**:

- ✓ Permite cambiar el **estilo del botón** entre **Vidrioso o Sólido**.
- ✓ Permite cambiar el **diseño de botón** entre **Normal, Botón con icono, Metro, Confirmar, Cancelar, Eliminar o Personalizado**.
- ✓ Permite establecer **esquinas redondeados al botón** (Solo para el estilo solido).
- ✓ Permite cambiar el **tamaño de borde del botón** (Solo para el estilo vidrioso)

Propiedades

`BorderRadius`	Obtiene o establece el radio del borde para aplicar esquinas redondeadas al botón.
`BorderSize`	Obtiene o establece el tamaño de borde del botón.
`Design`	Obtiene o establece el diseño del botón (Normal, Botón de icono, Metro, Confirmar, Cancelar, Eliminar o personalizado).
`Style`	Obtiene o establece el estilo del botón (vidrioso o sólido).

Métodos

`ApplyAppearanceSettings()`	Establece los valores de la configuración de apariencia a las propiedades del control.

`Button_HandleCreated(`
`object sender,`
`EventArgs e)`
	Invoca el método ApplyAppearanceSettings para aplicar la configuración de apariencia.

`IconButton()`	Establece las propiedades necesarias para el diseño del botón con icono.
`MetroButton()`	Establece las propiedades necesarias para el diseño metro del botón.
`NormalButton()`	Establece las propiedades necesarias para el diseño normal del botón.

Diseños de botón – Enumeración ButtonDesign

La enumeración diseños de botón (*ButtonDesing.cs*) define los diferentes diseños de apariencia para el botón, por ejemplo, metro, confirmar, normal y entre otros que se describen a continuación.

Campos

`Normal`	El botón tiene un **diseño común y plano**, generalmente el color de apariencia es el **mismo color al color de estilo** de la aplicación establecido por la configuración de apariencia.

`IconButton`	El botón tiene un **diseño con icono y plano**, generalmente el color de apariencia es el **mismo color al color de estilo** de la aplicación establecido por la configuración de apariencia.
`Metro`	El botón tiene un diseño **similar a los botones del menu inicio de Windows 8**, generalmente el color de apariencia es el **mismo color al color de estilo** de la aplicación establecido por la configuración de apariencia.
`Confirm`	El botón tiene un **diseño con icono y plano**, el color de apariencia es de **tono verde,** establecido en la lista de colores RJColors.
`Cancel`	El botón tiene un **diseño con icono y plano**, el color de apariencia es de **tono gris oscuro,** establecido en la lista de colores RJColors.
`Delete`	El botón tiene un **diseño con icono y plano**, el color de apariencia es de **tono rojo,** establecido en la lista de colores RJColors.
`Custom`	El botón tiene un **diseño con icono y plano o normal**, de acuerdo a su diseño anterior. Con esta opción puedes personalizar tanto el diseño y colores del botón (En este modo **no se aplica la configuración de apariencia** de la aplicación).

8.2.1.3. Clase RJ CheckBox

Esta clase **hereda** de la clase **CheckBox** de la librería **Windows.Forms.**

Implementa muchas propiedades de personalización, como cambiar el estilo del control y personalizar el color de apariencia, por ejemplo, el color de fondo, borde, icono, estado verificado, etc. Sin embargo, para poder cambiar el color de apariencia la propiedad Customizable debe estar establecido en verdadero. Este control no soporta tener texto.

Propiedades

`BackColor`	Obtiene o establece el color de fondo.
`BorderColor`	Obtiene o establece el color de borde.
`BorderSize`	Obtiene o establece el tamaño de borde.
`Check`	Obtiene o establece un valor que indica si el checkbox está marcado o no.
`CheckedColor`	Obtiene o establece el color de fondo o borde cuando el botón está marcado.
`Customizable`	Obtiene o establece si los colores de apariencia del control es personalizable, caso contrario, el color de apariencia es establecido por la configuración de apariencia.

`IconColor`	Obtiene o establece el color del icono de verificación.
`IconSize`	Obtiene o establece el tamaño del icono de verificación.
`MouseDownColor`	Obtiene o establece el color de fondo cuando se hace clic en el control.
`MouseOverColor`	Obtiene o establece el color de fondo cuando el mouse pasa sobre el control.
`Style`	Obtiene o establece el estilo de apariencia.

Métodos

`ApplyAppearanceSettings()`	Establece los valores de la configuración de apariencia a las propiedades del control.
`CheckBox_CheckChanged(object sender, EventArgs e)`	Se encarga de establecer o quitar el icono de verificación cuando el control cambia de estado o valor.
`CheckBox_Resized(object sender, EventArgs e)`	Se encarga de mantener el tamaño establecido o establecer un tamaño fijo cuando la propiedad AutoSize esté establecido en verdadero.

8.2.1.4. Clase RJ Label

Esta clase **hereda** de la clase **Label** de la librería **Windows.Forms.**

Este control implementa **dos propiedades de apariencia** que permite **cambiar el estilo** (**Normal, Titulo, Descripción, etc.**) y si es una **etiqueta de enlace**.

Propiedades

`LinkLabel`	Obtiene o establece si la etiqueta es un enlace (Si es verdadero la etiqueta cambia de puntero y color de texto cuando el mouse pasa sobre el)
`Style`	Obtiene o establece el estilo de apariencia.

Métodos

`ApplyAppearanceSettings()`	Establece los valores de la configuración de apariencia a las propiedades del control.
`Label_MouseEnter(object sender, EventArgs e)`	Se encarga de resaltar el color del texto si la etiqueta de tipo enlace.
`Label_MouseLeave(object sender, EventArgs e)`	Se encarga de volver a establecer el color de texto original si la etiqueta es de tipo enlace.

8.2.1.5. Clase RJ RadioButton

Esta clase **hereda** de la clase **RadioButton** de la librería **Windows.Forms.**

Este control **anula completamente el evento de pintura** y se dibuja un **nuevo diseño de botón de radio** con los **colores asignados en la configuración de apariencia.**

Campos

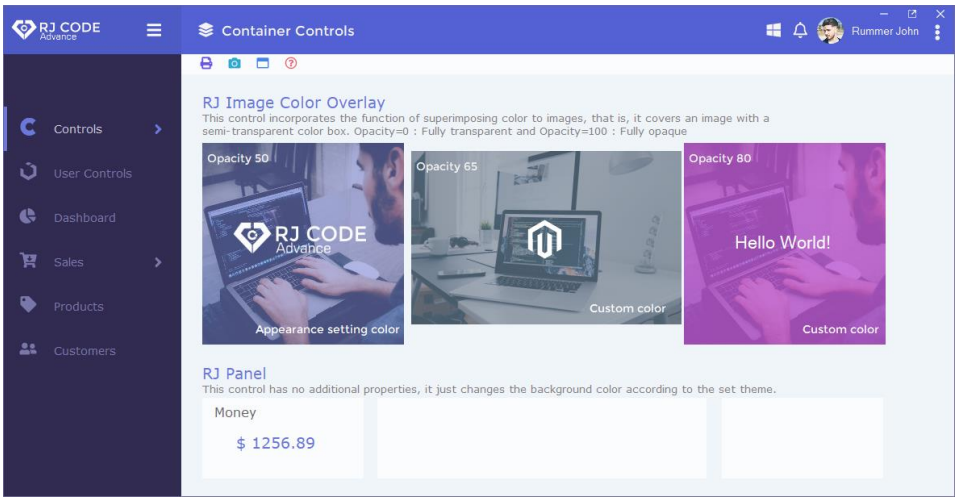
`backgroundColor`	Obtiene o establece el color de fondo.
`borderColor`	Obtiene o establece el color de borde.
`checkedColor`	Obtiene o establece el color del punto selector en su estado marcado.

Métodos

`ApplyAppearanceSettings()`	Establece los valores de la configuración de apariencia a las propiedades del control.
`OnPaint(PaintEventArgs e)`	Se encarga de dibujar la interfaz gráfica del control desde cero y el texto.
`rbCheckedChanged(object sender, EventArgs e)`	Se encarga de volver a dibujar la interfaz gráfica cuando el control cambia de valor: estado marcado o estado no marcado.
`rbResize(object sender, EventArgs e)`	Se encarga de calcular y establecer el tamaño adecuado del control cada vez que ocurra algún cambio de tamaño.
`rbTextChanged(object sender, EventArgs e)`	Se encarga de volver a dibujar el texto cuando cambia el texto del control.

8.2.2. Controles contenedores

Los controles contenedores son aquellos controles que **pueden tener otros controles en su interior**, por ejemplo en la librería de Windows Forms tenemos: GroupBox, Panel, TabControl y entre otros. En este proyecto están **disponible dos controles contenedores**.



8.2.2.1. Diagrama de clases

Diagrama de clases contraído

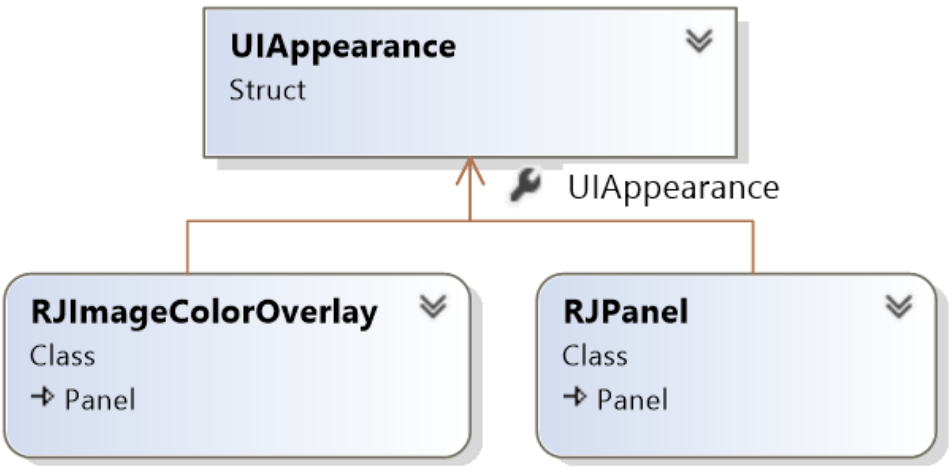
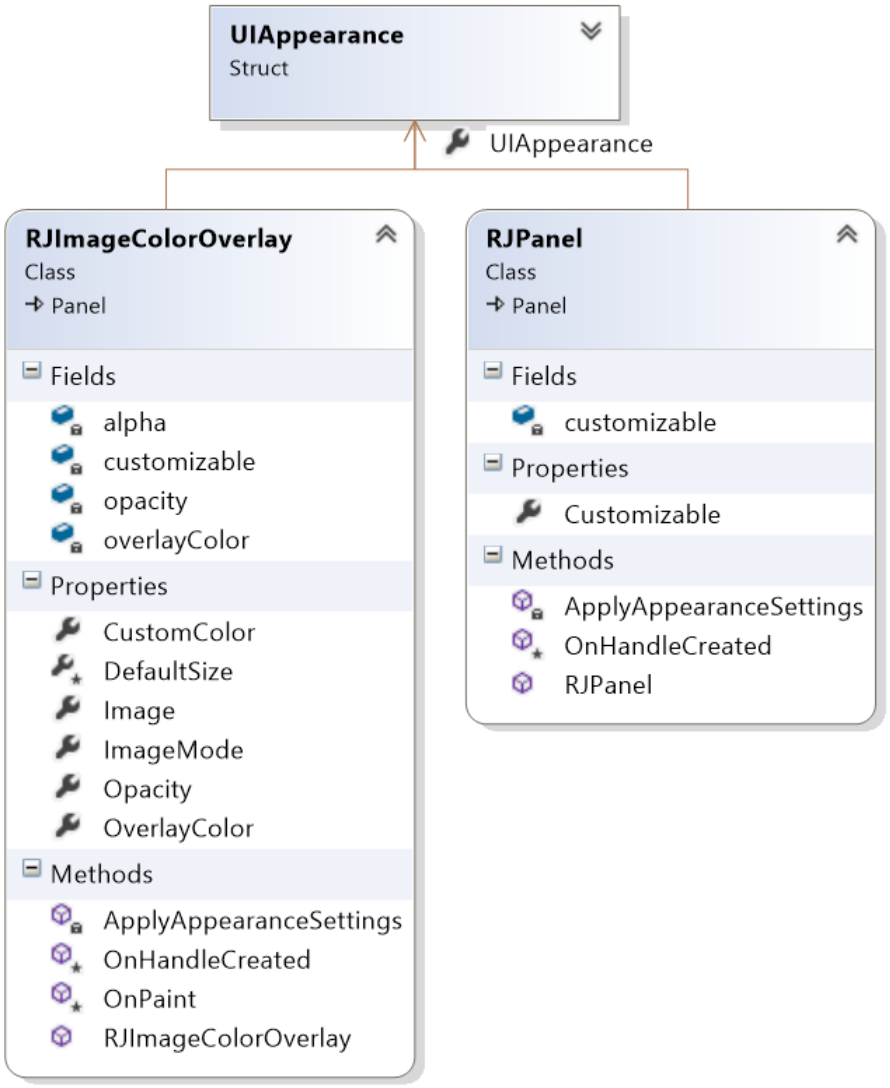


Diagrama de clases expandido



8.2.2.2. Clase RJ ImageColorOverlay

Esta clase **hereda** de la clase **Panel** de la librería **Windows.Forms.**

Este control incorpora la función de **superposición de color a una imagen**, es decir, cubre una imagen con una caja de **color semitransparente**, puedes controlar la intensidad de transparencia, donde **opacidad 0 es totalmente transparente** y **opacidad 100 es Totalmente opaco o Solido**. Además que **permite agregar otros controles en su interior** (Como se muestra en la captura de pantalla).

Propiedades

`Customizable`	Obtiene o establece si los colores de apariencia del control es personalizable, caso contrario, el color de apariencia es establecido por la configuración de apariencia.
`Image`	Obtiene o establece la imagen.
`ImageMode`	Obtiene o establece el diseño de la imagen (Centrado, Tramo, Loseta o enfocado).
`Opacity`	Obtiene o establece la intensidad de transparencia.
`OverlayColor`	Obtiene o establece el color de superposición.

Métodos

`ApplyAppearanceSettings()`	Establece los valores de la configuración de apariencia a las propiedades del control.
`OnPaint(PaintEventArgs e)`	Se encarga de dibujar la caja de color semitransparente.
`OnHandleCreated(EventArgs e)`	Invoca el método ApplyAppearanceSettings.

8.2.2.3. Clase RJ Panel

Esta clase **hereda** de la clase **Panel** de la librería **Windows.Forms**.

Este control no tiene propiedades adicionales de personalización, simplemente **establece el color de fondo según del tema establecido** por la configuración de apariencia.

Propiedades

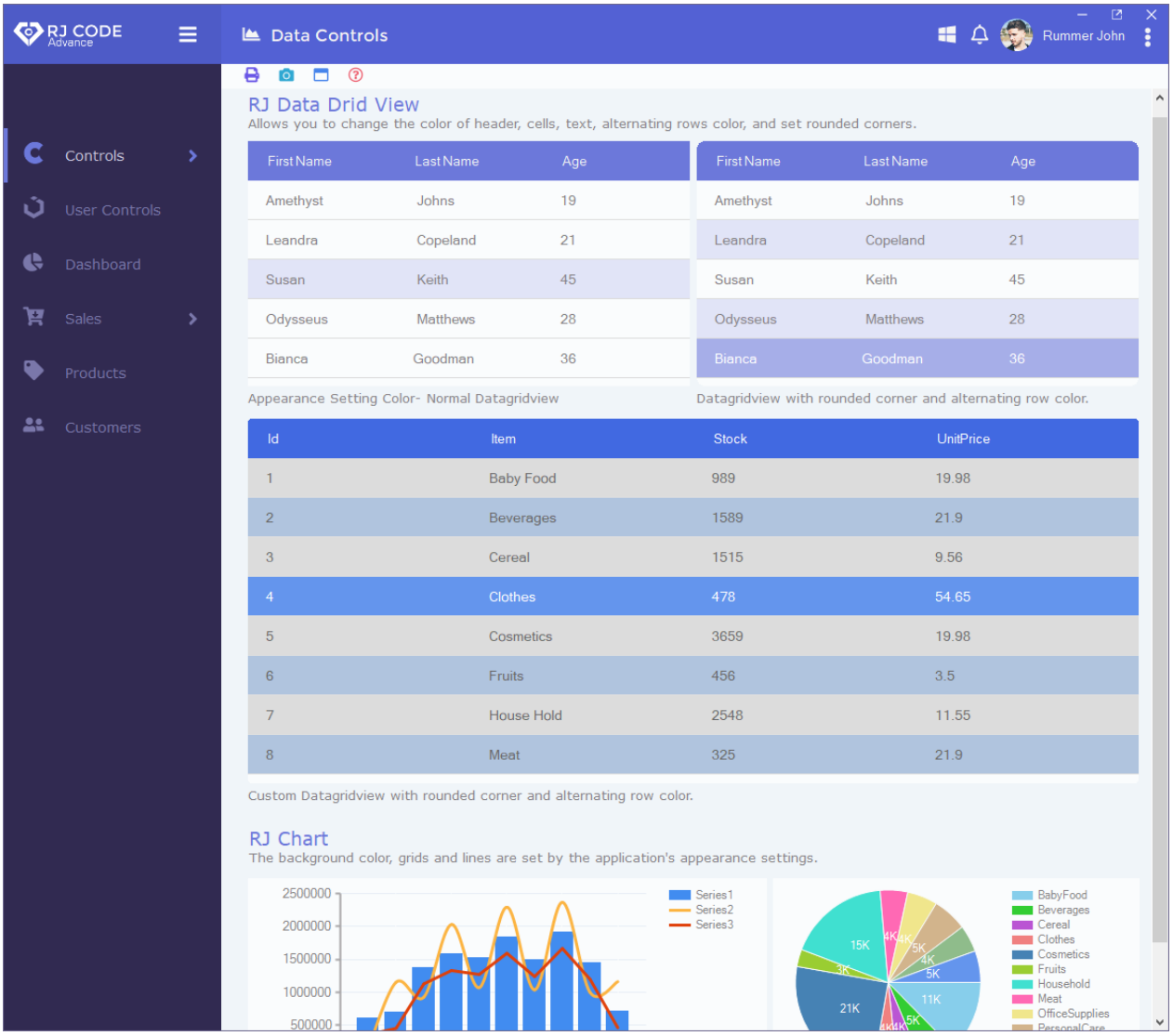
`Customizable`	Obtiene o establece si el color de fondo del control es personalizable, caso contrario, el color de fondo es establecido por la configuración de apariencia.

Métodos

`ApplyAppearanceSettings()`	Establece los valores de la configuración de apariencia a las propiedades del control.
`OnHandleCreated(EventArgs e)`	Invoca el método ApplyAppearanceSettings.

8.2.3. Controles de datos

Este tipo de control permite **representar datos en la interfaz gráfica desde una fuente de datos**.



8.2.3.1. Diagrama de clases

Diagrama de clases contraído

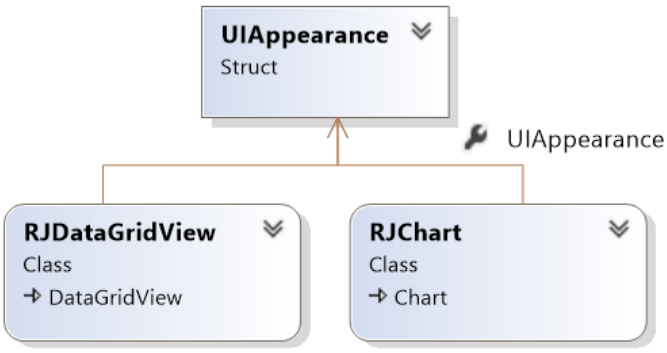
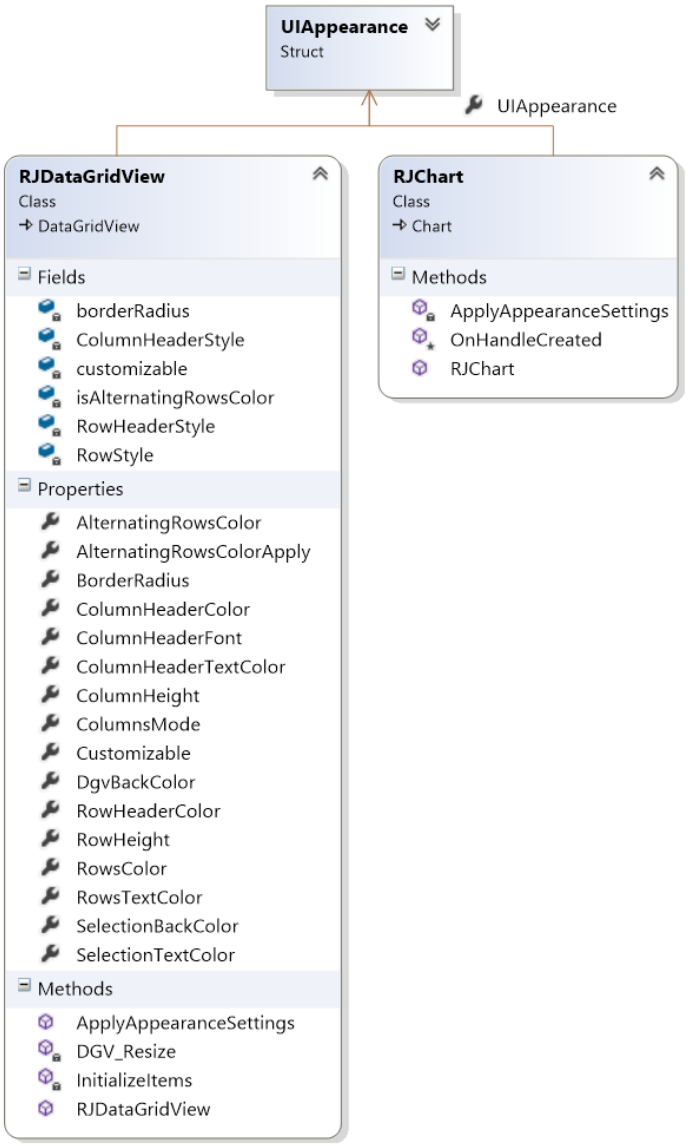


Diagrama de clases expandido



8.2.3.2. Clase RJ DataGridView

Esta clase **hereda** de la clase **Datagridview** de la librería **Windows.Forms**.

Este control implementa varias **propiedades de personalización**, como: **cambiar el color** de la cabecera, filas, grillas, fondo y entre otros, también de poder habilitar el **color de filas alternas** y establecer un color para ello. Además de poder aplicar **esquinas redondeadas** al control. Por defecto estos colores son establecidos por la configuración de apariencia, si deseas cambiarlos establece la propiedad Customizable en verdadero.

Propiedades

`AlternatingRowsColor`	Obtiene o establece el color para las filas alternas.
`AlternatingRowsColorApply`	Obtiene o establece si aplicar o no el color de filas alternas.
`BordeRadius`	Obtiene o establece el radio para las esquinas redondeadas.
`ColumnHeaderColor`	Obtiene o establece el color de la cabecera de las columnas.
`ColumnHeaderFont`	Obtiene o establece la fuente de la cabecera de las columnas.

`ColumnHeaderTextColor`	Obtiene o establece el color de texto de la cabecera.
`ColumnHeaderHeight`	Obtiene o establece el alto de la cabera de las columnas.
`ColumnsMode`	Obtiene o establece el modo de tamaño automático de las columnas.
`Customizable`	Obtiene o establece si el color de fondo del control es personalizable, caso contrario, el color de fondo es establecido por la configuración de apariencia.
`DgvBackColor`	Obtiene o establece el color de fondo.
`RowHeaderColor`	Obtiene o establece el color de fondo de la cabera de filas.
`RowsColor`	Obtiene o establece el color de las filas.
`RowsTextColor`	Obtiene o establece el color de texto de las filas.
`SelectionBackColor`	Obtiene o establece el color de fondo de la fila seleccionada.
`SelectionTextColor`	Obtiene o establece el color de texto de la fila seleccionada.

Métodos

`ApplyAppearanceSettings()`	Establece los valores de la configuración de apariencia a las propiedades del control.
`DGV_Resize(object sender, EventArgs e)`	Se encarga de aplicar la configuración de apariencia en tiempo de ejecución y validar el alto mínimo de la cabecera de la columna.

8.2.3.3. Clase RJ Chart

Esta clase **hereda** de la clase **Chart** de la librería **Windows.Forms**.

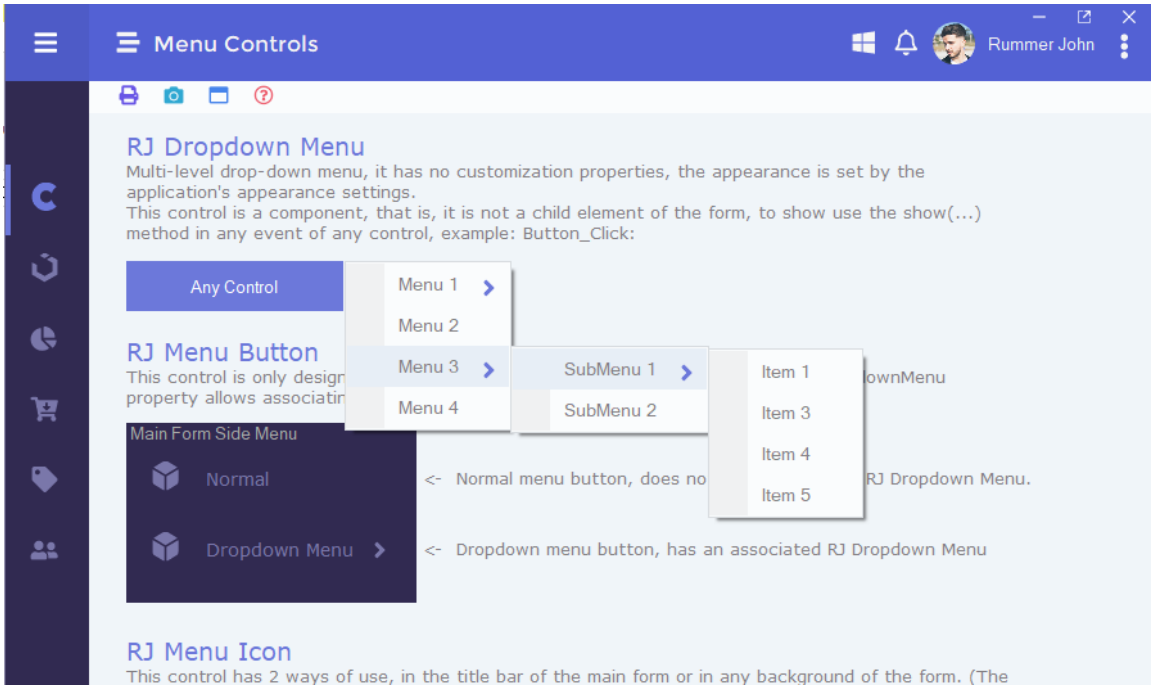
Este control no implementa propiedades de personalización de apariencia, **el color de fondo, grillas, líneas y texto es implementado por la configuración de apariencia**. Sin embargo puedes modificar las otras propiedades originales del control Chart, como **cambiar el color de paleta, cambiar el tamaño de las líneas**, y entre otros.

Métodos

`ApplyAppearanceSettings()`	Establece los valores de la configuración de apariencia a las propiedades del control.
`OnHandleCreated( EventArgs e)`	Se encarga de aplicar la configuración de apariencia en tiempo de ejecución o diseño.

8.2.4. Controles de menú

Los controles de menú **permiten la organización jerárquica de elementos** de manera multinivel, y/o pueden ser usados en el **menú lateral y en la barra de título** del formulario principal.



8.2.4.1. Diagrama de clases

Diagrama de clases contraído

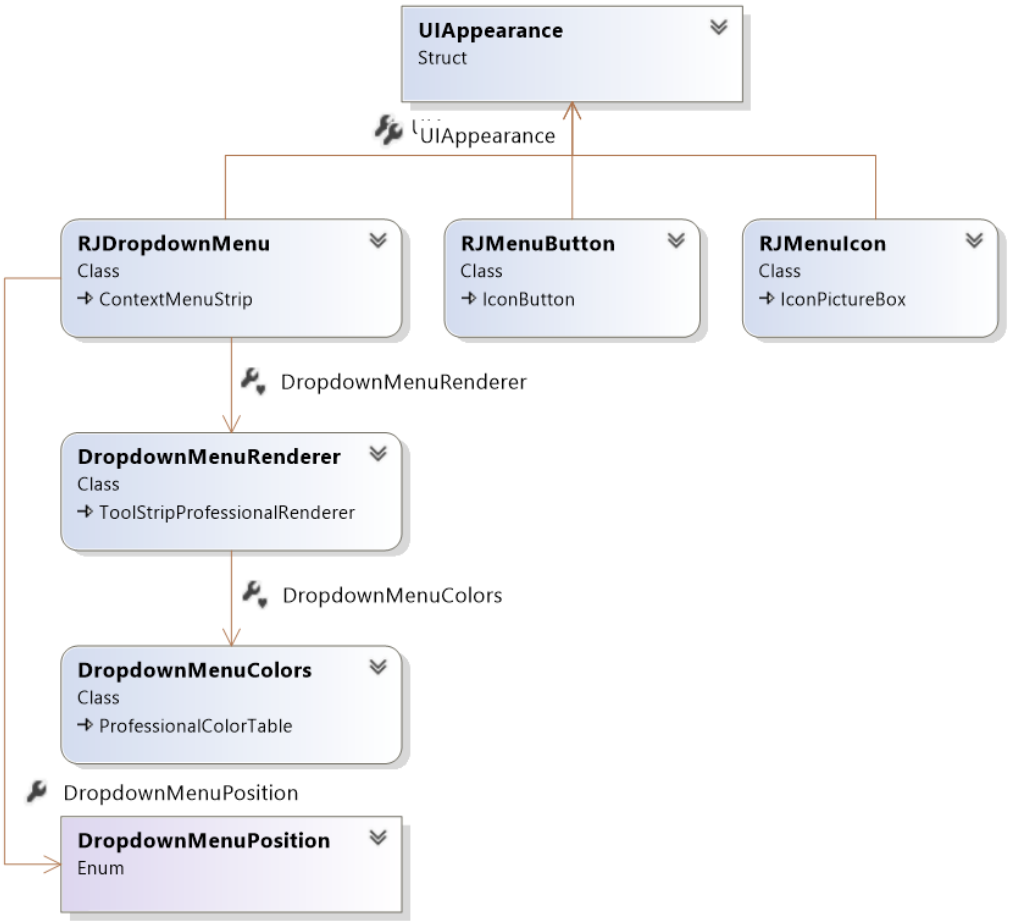
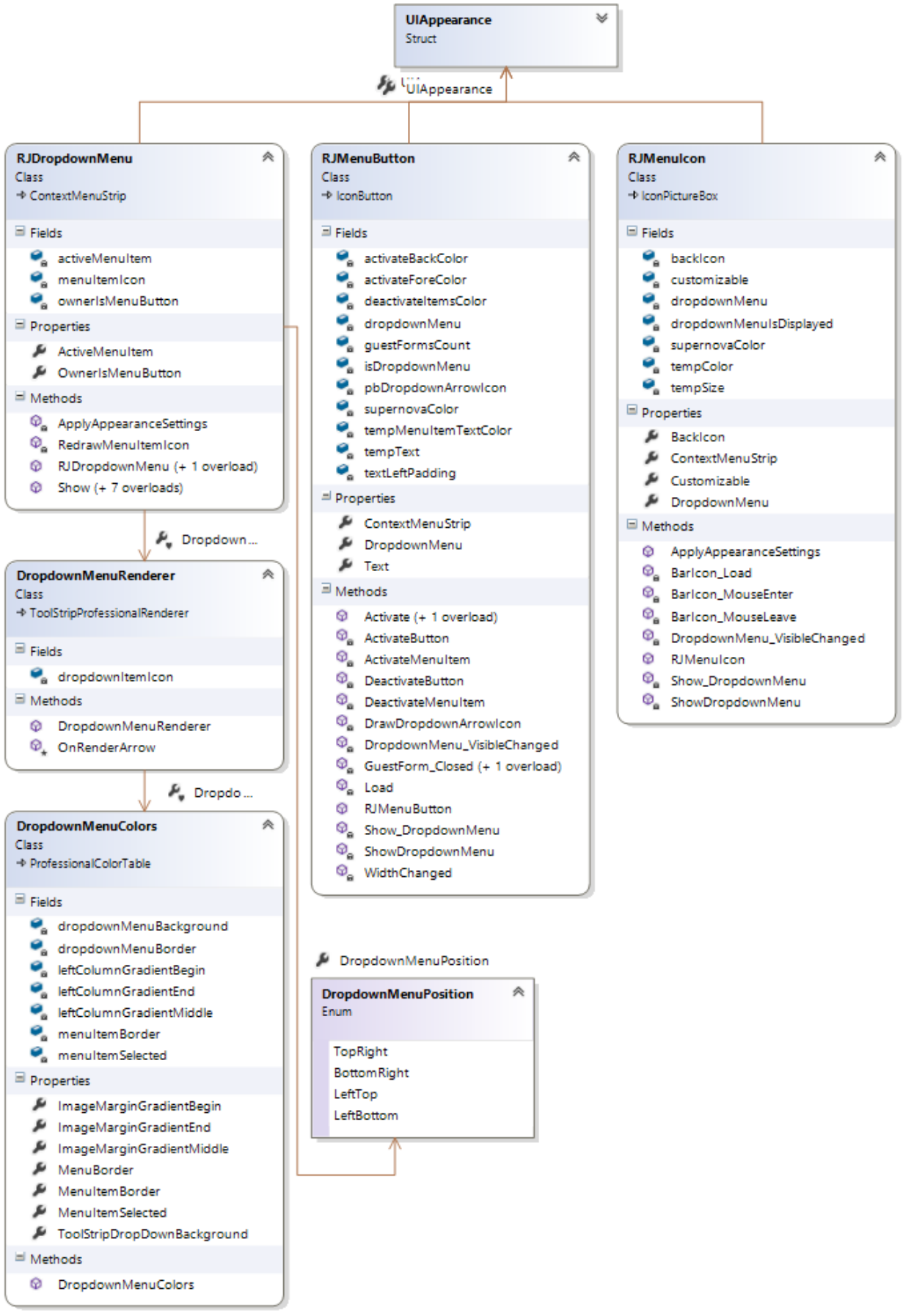


Diagrama de clases expandido



8.2.4.2. Clase RJ DropdownMenu

Esta clase **hereda** de la clase **ContextMenuStrip** de la librería **Windows.Forms**.

Este control de **menú desplegable**, permite tener **varios elementos multinivel** (Organizados jerárquicamente), no expone propiedades de personalización. **El color de apariencia se establece según el tema y estilo establecido** por la configuración de apariencia en la aplicación, para ello **es necesario las clases DropdownMenuRendereder y DropdownMenuColors.**

Tener en cuenta que este control también **es un componente**, asegúrate de **desecharlo manualmente** si la instanciaste por código, o puedes **agregarlo a un objeto de tipo componente** para luego desecharlo. Si arrastraste el control al formulario, **se instancia automáticamente mediante su constructor que implementa *IContainer***, por lo tanto se desechará junto con el formulario y los otros componentes donde no es necesario desecharlo manualmente (Ver el método **Dispose(bool disposing)** de cualquier **formulario.designer.cs**), para obtener más información, **ver el punto 8.4** (Componentes).

Propiedades

`ActiveMenuItem`	Obtiene o establece si el menú desplegable tiene un elemento de menú activo.
`OwnerIsMenuItem`	Obtiene o establece si el propietario del menú desplegable es un **botón de menú** (RJMenuButton).

Métodos

`ApplyAppearanceSettings()`	Establece los valores de la configuración de apariencia a las propiedades del control.
`RedrawMenuItemIcon(Image itemImage)`	Se encarga de dibujar el icono/imagen del elemento de menú con un tamaño adecuado y centrado.
`RJDropdownMenu()`	Constructor sin parámetros, al usar este constructor asegúrate de desechar el control manualmente.
`RJDropdownMenu(IContainer container)`	Constructor con parámetro IContainer, **este constructor es invocado automáticamente en el diseñador del formulario** cuando el control es arrastrado desde la caja de herramientas al formulario. Este constructor **asegura que el objeto se elimine correctamente**, ya que no es un elemento secundario del formulario.
`Show(Control ownerControl, DropdownMenuPosition position)`	Permite **mostrar y posicionar** (ver la enumeración ***DropdownMenuPosition.cs***) el menú desplegable más rápidamente.
`Show(+6 overloads)`	Permite mostrar el menú desplegable.

Clase DropdownMenuRenderer

Esta clase hereda de la clase **ToolStripProfessionalRenderer** de la librería **Windows.Forms.**

Esta clase controla la funcionalidad del dibujo, mediante la aplicación de una paleta personalizada y un estilo optimizado.

Campos

`dropdownItemIcon`	Obtiene o establece el icono de flecha del elemento de menú desplegable.

Métodos

`DropdownMenuRenderer()`	Se encarga de enviar el objeto tabla de colores profesional que se utiliza para dibujar el menú desplegable.
`OnRenderArrow(ToolStripArrowRenderEventArgs e)`	Se encarga de dibujar el icono de flecha desplegable de un elemento de menú en caso que sea padre de otros elementos de menú.

Clase DropdownMenuColors

Esta clase hereda de la clase **ProfessionalColorTable** de la librería **Windows.Forms.**

Esta clase **establece los colores de apariencia del menú desplegable** a partir de la configuración de apariencia de la aplicación.

Campos

`dropdownMenuBackground`	Obtiene o establece el color de fondo.
`dropdownMenuBorder`	Obtiene o establece el color de borde.
`leftColumnGradientBegin`	Obtiene o establece el color de inicio de la columna izquierda del menú desplegable.
`leftColumnGradientMiddle`	Obtiene o establece el color del centro de la columna izquierda del menú desplegable.
`leftColumnGradientEnd`	Obtiene o establece el color de la parte final de la columna izquierda del menú desplegable.
`menuItemSelected`	Obtiene o establece el color de fondo del elemento de menú seleccionado.
`menuItemBorder`	Obtiene o establece el color de borde del elemento de menú.

Métodos

`DropdownMenuColors(bool menuButtonOwner)`	Se encarga de inicializar los campos de apariencia según la configuración de apariencia de la aplicación.

8.2.4.3. Clase RJ MenuButton

Esta clase hereda de la clase **IconButton** de la librería **FontAwesome.Sharp.**

Este es un control especial que **solo está diseñado para ser utilizado en el menú lateral del formulario principal.** Puede funcionar como un botón de **menú normal o como un botón de menú desplegable**, para esto **es necesario agregar un menú desplegable** (RJDropdownMenu) desde la **propiedad DropdownMenu** de este control, el evento de clic se creará automáticamente para mostrar el menú desplegable.

Tiene 2 métodos esenciales de apariencia:

- ✓ Como un **botón de menú normal**; permite **asociar un formulario y el botón se activa**/resalta hasta que se cierra el formulario (Ver el método *Activate(RJChildForm)*).
- ✓ Como un **botón de menú desplegable**; permite **asociar muchos formularios**, el **botón y el elemento de menú se activan**/resaltan hasta cerrar el formulario (Ver método *Activate (RJChildForm, ToolStripMenuItem*))

Propiedades

`DropdownMenu`	Obtiene o establece un control ***RJDropdownMenu*** al botón de menú, el evento clic se crea automáticamente para mostrar el menú desplegable.
`Text`	Obtiene o establece el texto del control, su función especial es agregar un relleno antes del texto para mantener cierta distancia entre el icono y el texto.

Métodos

`Activate(RJChildForm guestForm)`	Se encarga de asociar un formulario secundario y activar el botón de menú.
`Activate(RJChildForm guestForm, ToolStripMenuItem menuItem)`	Se encarga de asociar un formulario secundario y activar el botón de menú y el elemento de menú propietario del formulario.
`ActivateButton()`	Es responsable de activar o resaltar el botón de menú.
`ActivateMenuItem(RJChildForm guestForm, ToolStripMenuItem menuItem)`	Es responsable de activar o resaltar el elemento de menú del menú desplegable.
`DeactivateButton()`	Desactiva el botón. La apariencia del botón se restablece.
`DeactivateMenuItem( ToolStripMenuItem menuItem)`	Desactiva el elemento de menú. La apariencia se restablece.
`DrawDropdownArrowIcon(bool expandedMenu)`	Se encarga de dibujar el icono de flecha del botón de menú desplegable en su estado expandido o contraído.
`DropdownMenu_VisibleChanged( object sender, EventArgs e)`	Activa el botón de menú desplegable cuando se muestra el menú desplegable asociado, además se

	encarga de invocar el método DrawDropdownArrowIcon().
`GuestForm_Closed(object sender, FormClosedEventArgs e)`	Se encarga de desactivar el botón de menú cuando el formulario es cerrado.
`GuestForm_Closed(object sender, FormClosedEventArgs e, ToolStripMenuItem menuItem)`	Se encarga de desactivar el elemento de menú cuando el formulario es cerrado, y desactivar el botón de menú cuando todos los formularios asociados a este son cerrados.
`MB_HandleCreated(object sender, EventArgs e)`	Se encarga de almacenar el texto del botón de menú en un campo temporal.
`ShowDropdownMenu()`	Es responsable de **mostrar el menú desplegable** (***RJDropdownMenu***)
`DropdownMenuButton_Click(object sender, EventArgs e)`	Invoca el método ShowDropdownMenu cuando se hace clic en el control.
`WidthChanged(object sender, EventArgs e)`	Es responsable de ocultar o mostrar el texto del botón de menú cuando el menú lateral del formulario principal se contrae o expande.

8.2.4.4. Clase RJ MenuIcon

Esta clase hereda de la clase **IconPictureBox** de la librería **FontAwesome.Sharp.**

Este control está **diseñado principalmente en la barra de título del formulario principal,** sin embargo, puede deshabilitarlo estableciendo la **propiedad BackIcon en verdadero.**

Al igual que el control RJMenuButton, este control puede funcionar como un icono de **menú normal o como un icono de menú desplegable**, para esto **es necesario agregar un menú desplegable** (RJDropdownMenu) desde la **propiedad DropdownMenu** de este control, el evento de clic se creará automáticamente para mostrar el menú desplegable.

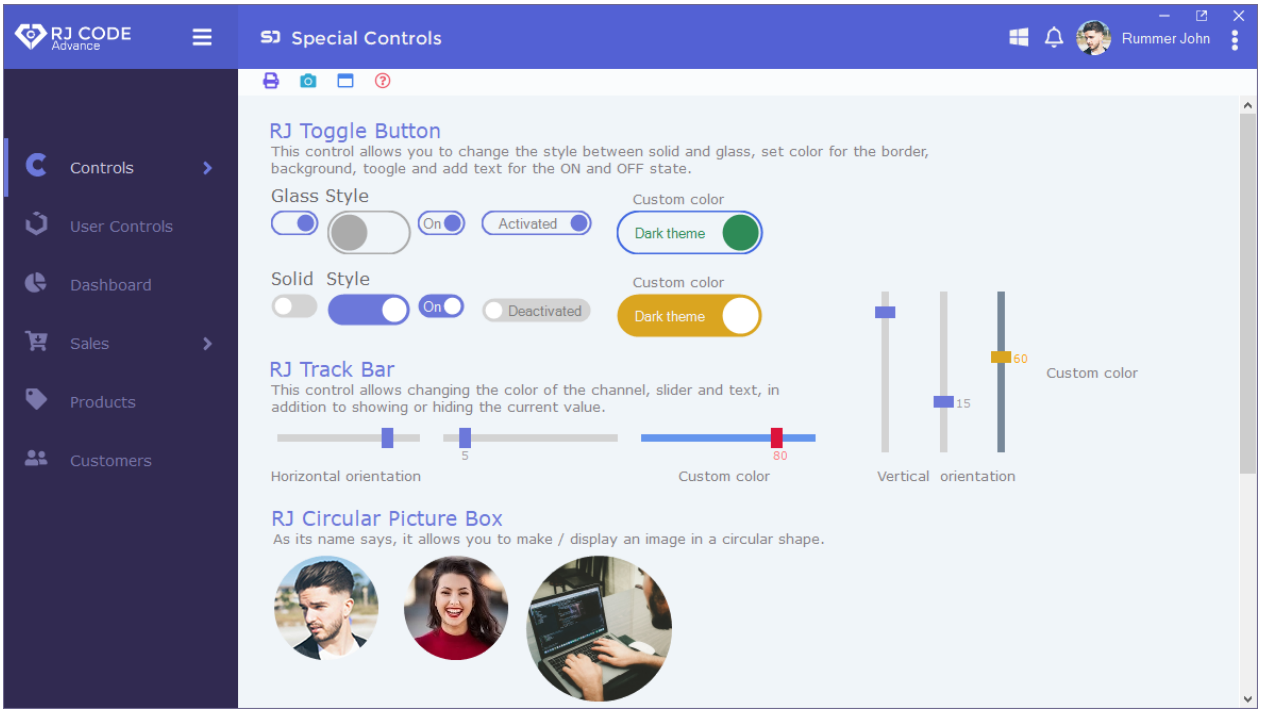
Permite cambiar el color de icono y fondo si establece la propiedad Customizable en verdadero.

Propiedades

`BackIcon`	Obtiene o establece si el control es un icono de menú de la barra de título del formulario principal (**False**), o es un icono de menú en cualquier área de cliente del formulario (**True**).
`Customizable`	Obtiene o establece si el color de fondo e icono del control es personalizable.
`DropdownMenu`	Obtiene o establece un control ***RJDropdownMenu*** al botón de menú, el evento clic se crea automáticamente para mostrar el menú desplegable.

8.2.5. Controles especiales

Son controles que no pertenecen a ninguna de las categorías anteriores o **no se encuentra en los controles convencionales** de la librería **Windows.Forms.**



8.2.5.1. Diagrama de clases

Diagrama de clases contraído

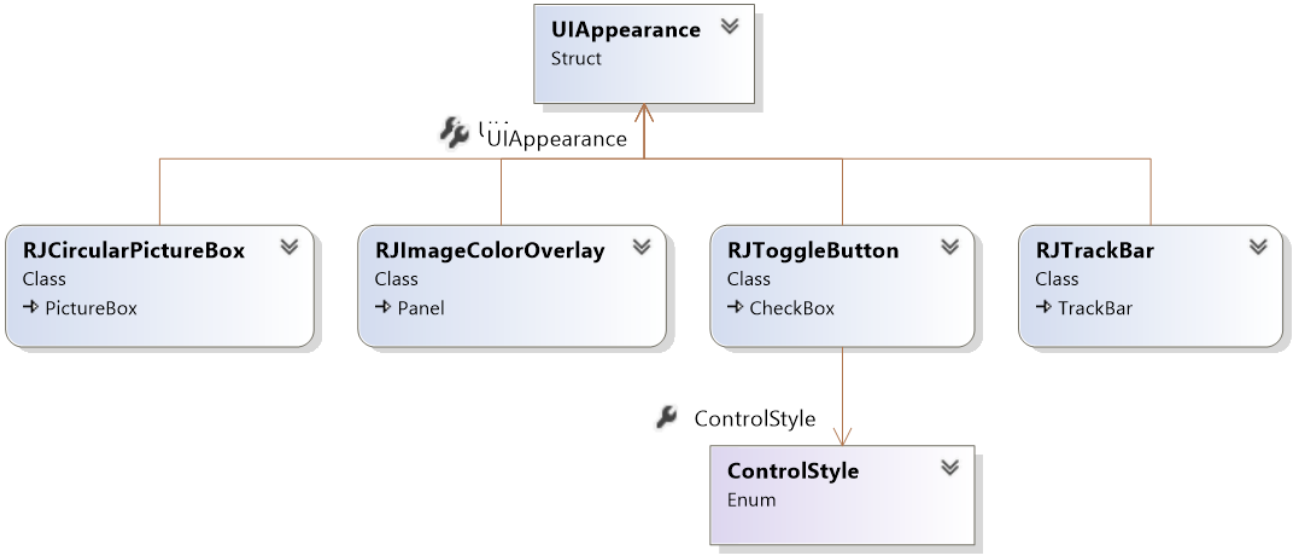
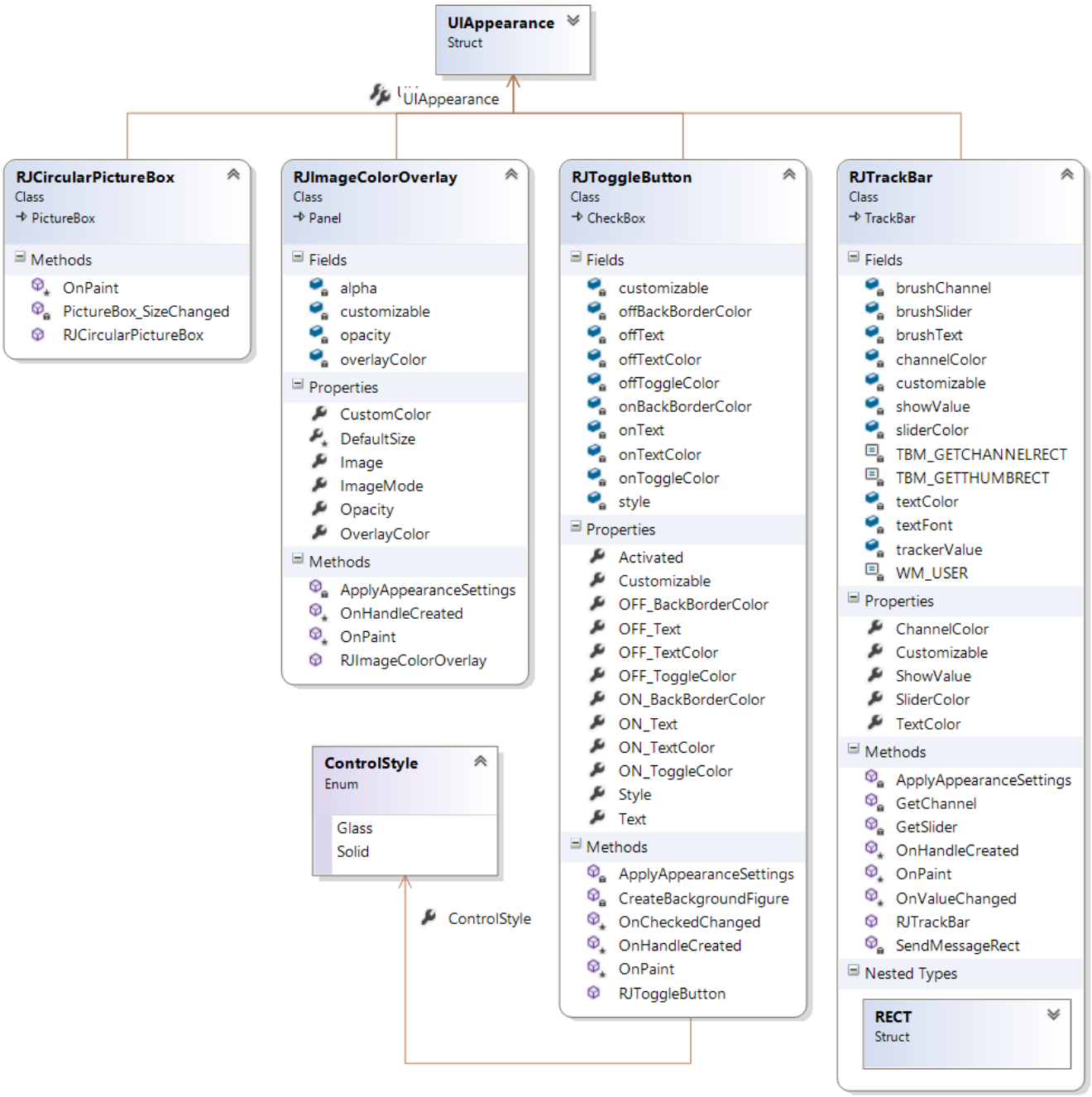


Diagrama de clases expandido



8.2.5.2. Clase RJ CircularPictureBox

Esta clase hereda de la clase **PictureBox** de la librería **Windows.Forms.**

Este control permite hacer y mostrar una **imagen** en **forma circular.**

Métodos

`OnPaint(PaintEventArgs e)`	Es responsable de crear el control en forma circular.
`PictureBox_SizeChanged(object sender, EventArgs e)`	Se encarga de mantener el circulo con el mismo radio (Circulo perfecto- mismo alto y ancho).

8.2.5.3. Clase RJ ToggleButton

Esta clase hereda de la clase **CheckBox** de la librería **Windows.Forms.**

Este control anula completamente el evento de pintura **y se dibuja un nuevo diseño del control**. Implementa una gran **variedad de propiedades de personalización de apariencia**. Por defecto es establecido de acuerdo a la configuración de apariencia de la aplicación, puedes cambiarlo estableciendo la propiedad Customizable en verdadero.

Propiedades

`Activated`	Obtiene o establece un valor que indica si el control ToggleButton está activado o desactivado (Encendido o apagado).
`Customizable`	Obtiene o establece si los colores de apariencia es personalizable.
`OFF_BackBorderColor`	Obtiene o establece el color de fondo o borde del estado desactivado.
`OFF_Text`	Obtiene o establece el texto del estado desactivado.
`OFF_TextColor`	Obtiene o establece el color de texto del estado desactivado.
`OFF_ToggleColor`	Obtiene o establece el color de la palanca del estado desactivado.
`ON_BackBorderColor`	Obtiene o establece el color de fondo o borde del estado activado.
`ON_Text`	Obtiene o establece el texto del estado activado.
`ON_TextColor`	Obtiene o establece el color de texto del estado activado.
`ON_ToggleColor`	Obtiene o establece el color de la palanca del estado activado.
`Style`	Obtiene o establece el estilo del control (Solido o vidrioso).

Métodos

`ApplyAppearanceSettings()`	Establece la configuración de apariencia de la aplicación en las propiedades de apariencia del control.
`CreateBackgroundFigure()`	Responsable de dibujar el fondo o borde con extremos redondeados del control ToggleButton.
`OnCheckedChanged(EventArgs e)`	Responsable de volver a dibujar el control si el estado o valor cambia (Estado encendido o Estado apagado).
`OnHandleCreated(EventArgs e)`	Se encarga de cargar y aplicar la configuración de apariencia cuando el identificador del control es creado.
`OnPaint(PaintEventArgs e)`	Responsable de dibujar el nuevo diseño del control.

8.2.5.4. Clase RJ TrackBar

Esta clase hereda de la clase **TrackBar** de la librería **Windows.Forms.**

Este control anula completamente el evento de pintura **y se dibuja un nuevo diseño del control**. Implementa **propiedades de personalización de apariencia**. Por defecto es establecido de acuerdo a la configuración de apariencia de la aplicación, puedes cambiarlo estableciendo la propiedad Customizable en verdadero.

Este control está **basado en el ejemplo** sugerido de Hans Passant.

Propiedades

`ChannelColor`	Obtiene o establece el color del canal del control.
`Customizable`	Obtiene o establece si los colores de apariencia es personalizable.
`ShowValue`	Obtiene o establece si la etiqueta de valor se muestra.
`SliderColor`	Obtiene o establece el color del control deslizante.
`TextColor`	Obtiene o establece el color de texto de la etiqueta de valor.

Métodos

`ApplyAppearanceSettings()`	Establece la configuración de apariencia de la aplicación en las propiedades de apariencia del control.
`GetChanel()`	Responsable de obtener el tamaño y ubicación del canal de la barra de seguimiento de Windows.
`GetSlider()`	Responsable de obtener el tamaño y ubicación del control deslizante de la barra de seguimiento de Windows.

`OnHandleCreated(EventArgs e)`	Se encarga de cargar y aplicar la configuración de apariencia cuando el identificador del control es creado.
`OnPaint(PaintEventArgs e)`	Responsable de dibujar el nuevo diseño del control.
`OnValueChanged(EventArgs e)`	Responsable de volver a dibujar el control si el valor cambia.
`SendMessageRect(IntPtr hWnd, int msg, IntPtr wParam, ref RECT lParam)`	Método externo, responsable de enviar los mensajes de Windows.

8.3. Controles compuestos (User Control)

Como ya mencioné anteriormente, un control compuesto **se crea mediante múltiples controles existentes** utilizando la **clase control de usuario** (UserControl), para ello puedes **agregar un UserControl desde el diseñador** y agregar controles, o puedes **heredar la clase UserControl** y agregar controles mediante código, **los controles constituyentes del control de usuario conservan toda su funcionalidad** donde puedes exponer y enlazar propiedades, además de asociar o **crear los eventos necesarios**.

La clase UserControl es básicamente un contenedor para otros controles y proporciona enrutamiento de teclado para los controles secundarios y permite que los controles secundarios funcionen como un grupo.

Recomiendo usar este método cuando es realmente necesario y como última opción, ya que es más pesado que un control extendido o personalizado.

Microsoft recomienda usar controles de usuario por el siguiente caso y también estoy de acuerdo con ello:

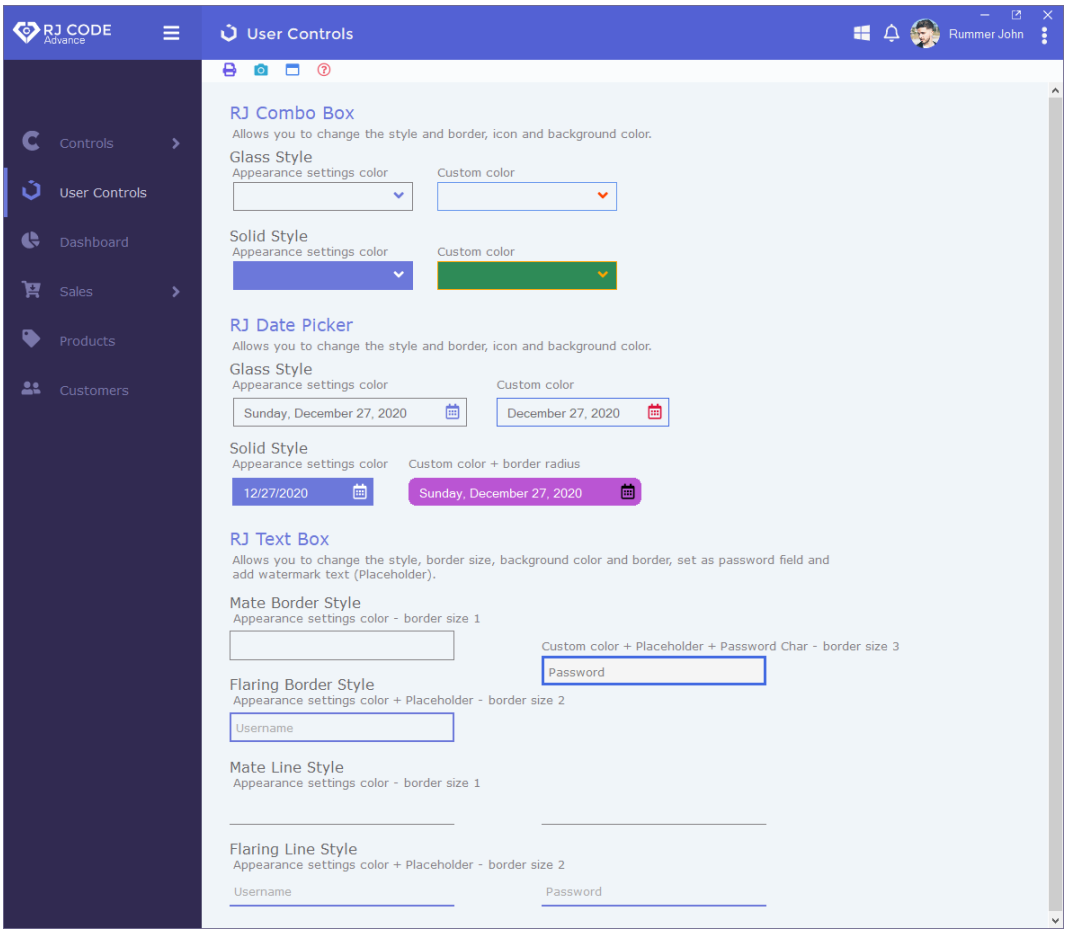
"Herede de la clase UserControl si:

- *Quiere combinar la funcionalidad de varios controles de Windows Forms en una sola unidad reutilizable."*

Bueno, en este proyecto solamente creé 3 controles de usuario:

1. Caja de combo
2. Selector de fecha
3. Caja de texto

A continuación se muestra una captura de pantalla de estos controles de usuario con todos los estilos y diseños disponibles:



8.3.1. Diagrama de clases

Diagrama de clases contraído

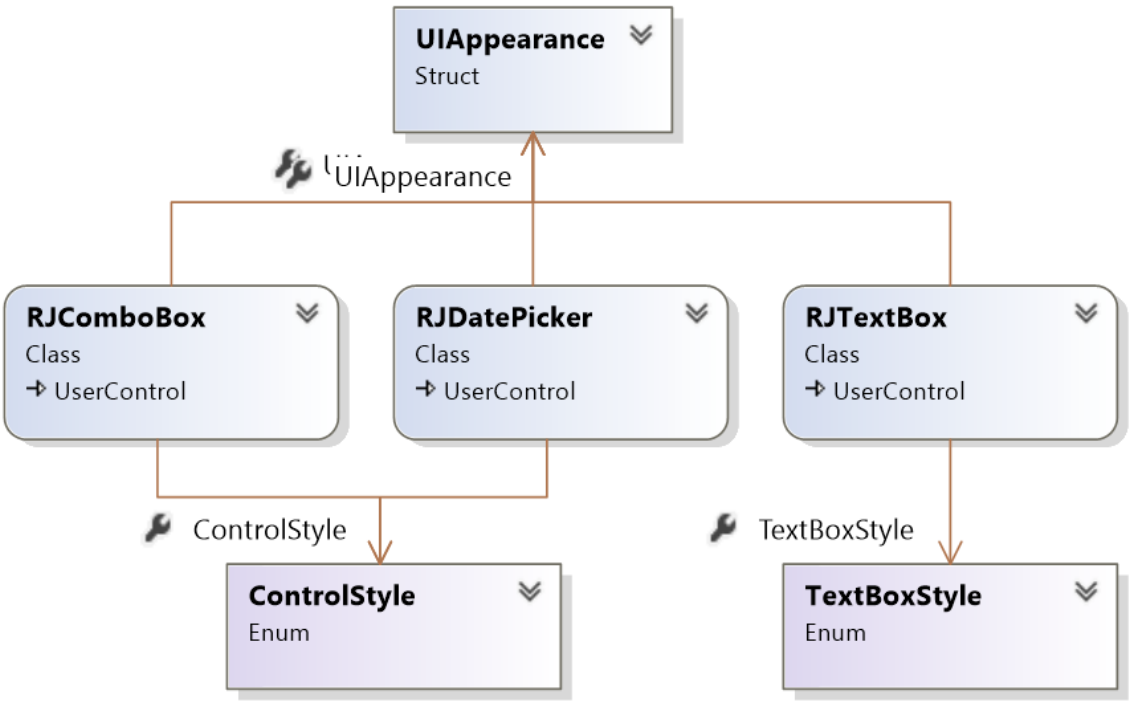
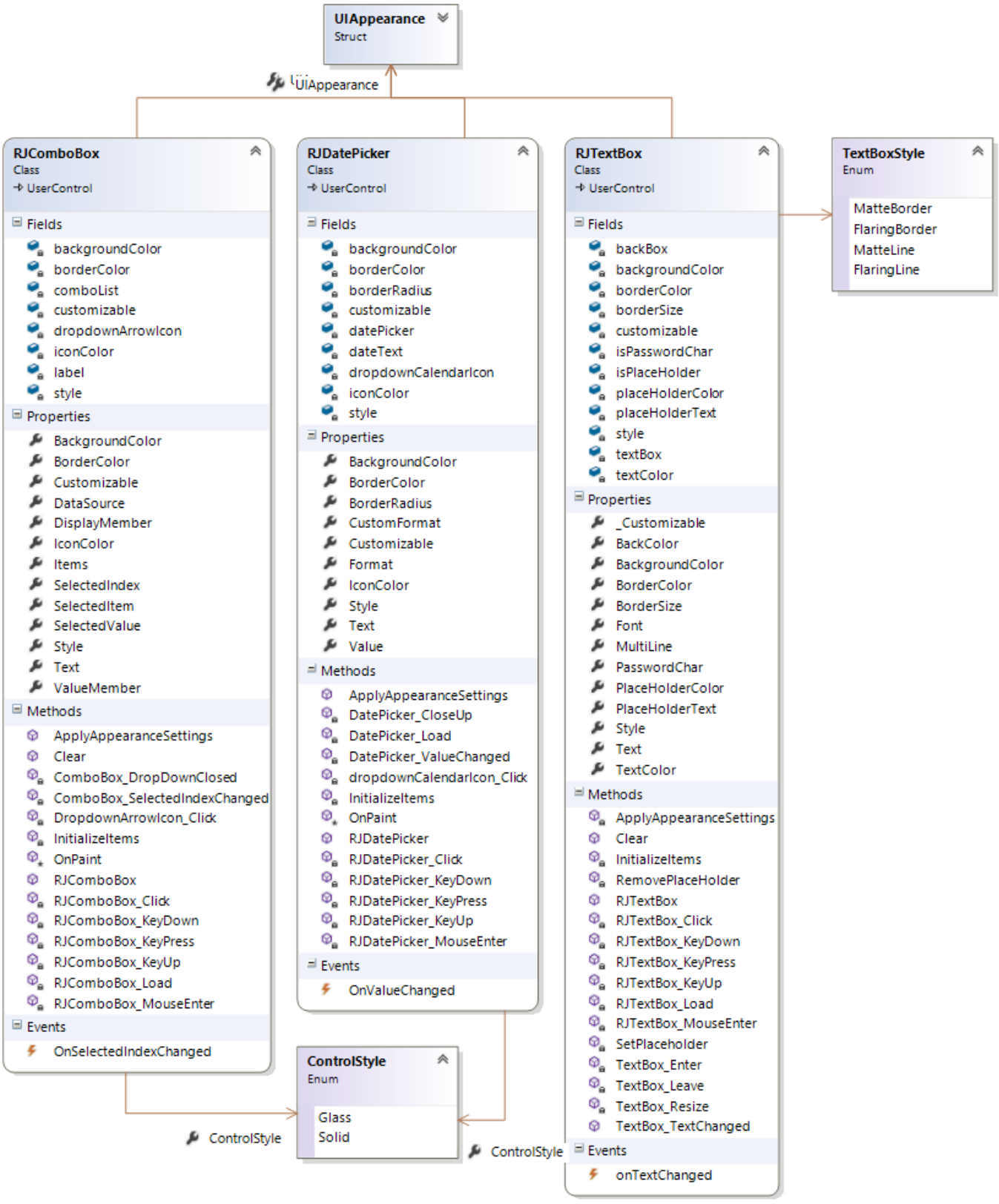


Diagrama de clases expandido



8.3.2. Clase RJ ComboBox

Esta clase hereda de la clase **UserControl** de la librería **Windows.Forms.**

Para crear el control RJComboBox, se agrega el control **ComboBox** y **Label** de la librería **Windows.Forms,** y también agrega el control **IconButton** de la librería **FontAwesome.Sharp.** Implementa **propiedades de personalización de apariencia** y **expone la mayor parte de las propiedades y eventos esenciales del control ComboBox** de Windows.

Propiedades

`BackgroundColor`	Obtiene o establece el color de fondo
`BorderColor`	Obtiene o establece el color de borde.
`Customizable`	Obtiene o establece si los colores de apariencia es personalizable.
`DataSource`	Obtiene o establece la fuente de datas para el combo box.
`DisplayMember`	Obtiene o establece la propiedad que se mostrará para este ListControl.
`IconColor`	Obtiene o establece el color de icono de la flecha despegable.
`Items`	Obtiene un objeto que representa la colección de elementos contenidos en este ComboBox.
`SelectedIndex`	Obtiene o establece el índice que especifica el elemento seleccionado actualmente.
`SelectedItem`	Obtiene o establece el elemento seleccionado actualmente en el ComboBox.
`SelectedValue`	Obtiene o establece el valor de la propiedad miembro especificada por la propiedad ValueMember.
`Style`	Obtiene o establece el estilo del control (Solido o Vidrioso)
`Text`	Obtiene o establece el texto del control de usuario.
`ValueMember`	Obtiene o establece la ruta de acceso de la propiedad que se utilizará como valor real para los elementos de ListControl.

Métodos

`ApplyAppearanceSettings()`	Establece la configuración de apariencia de la aplicación en las propiedades de apariencia del control.
`Clear()`	Elimina todos los elementos del ComboBox.

`ComboBox_DropDownClosed(object sender, EventArgs e)`	Responsable de restablecer la apariencia del control cuando se cierra la lista desplegable.
`ComboBox_SelectedIndexChanged(object sender, EventArgs e)`	Responsable de actualizar el texto del control cuando el índice de selección cambia.
`DropdownArrowIcon_Click(object sender, EventArgs e)`	Responsable de mostrar la lista desplegable del ComboBox y establecer la apariencia de estado activo.
`RJComboBox_Click(object sender, EventArgs e)`	Responsable de vincular el evento clic del control frontal constituyente al evento clic del control de usuario.
`RJComboBox_MouseEnter(object sender, EventArgs e)`	Responsable de vincular el evento MouseEnter del control frontal constituyente al evento MouseEnter del control de usuario.
`RJComboBox_KeyUp(object sender, KeyEventArgs e)`	Responsable de vincular el evento KeyUp del control frontal constituyente al evento KeyUp del control de usuario.
`RJComboBox_KeyPress(object sender, KeyPressEventArgs e)`	Responsable de vincular el evento KeyPress del control frontal constituyente al evento KeyPress del control de usuario.
`RJComboBox_KeyDown(object sender, KeyEventArgs e)`	Responsable de vincular el evento KeyDown del control frontal constituyente al evento KeyDown del control de usuario.

Eventos

`OnSelectedIndexChanged`	Evento por defecto del control de usuario recreando y vinculando al evento original SelectedIndexChanged del control ComboBox.

8.3.3. Clase RJ DatePicker

Esta clase hereda de la clase **UserControl** de la librería **Windows.Forms.**

Para crear el control RJDatePicker, se agrega el control **DateTimePicker** y **Label** de la librería **Windows.Forms,** y también agrega el control **IconButton** de la librería **FontAwesome.Sharp.** Implementa **propiedades de personalización de apariencia** y **expone la mayor parte de las propiedades y eventos esenciales del control DateTimePicker** de Windows.

Propiedades

`BackgroundColor`	Obtiene o establece el color de fondo
`BorderColor`	Obtiene o establece el color de borde.
`BorderRadius`	Obtiene o establece el radio para aplicar esquinas redondeadas al control.

`CustomFormat`	Obtiene o establece la cadena de formato de fecha personalizada.
`Customizable`	Obtiene o establece si los colores de apariencia es personalizable.
`Format`	Obtiene o establece el formato de la fecha que se muestra en el control.
`IconColor`	Obtiene o establece el color de icono.
`Style`	Obtiene o establece el estilo del control (Solido o Vidrioso)
`Text`	Obtiene el texto del control de usuario.
`Value`	Obtiene o establece el valor de fecha asignado al control.

Métodos

`ApplyAppearanceSettings()`	Establece la configuración de apariencia de la aplicación en las propiedades de apariencia del control.
`DatePicker_CloseUp(object sender, EventArgs e)`	Responsable de restablecer la apariencia del control cuando se cierra el calendario.
`DatePicker_ValueChanged(object sender, EventArgs e)`	Responsable de actualizar el texto del control cuando el valor del selector de fecha cambia.
`DropdownCalendarIcon_Click(object sender, EventArgs e)`	Responsable de mostrar el calendario del selector de fecha y establecer la apariencia de estado activo.
`RJDatePicker_Click(object sender, EventArgs e)`	Responsable de vincular el evento clic del control frontal constituyente al evento clic del control de usuario.
`RJDatePicker_MouseEnter(object sender, EventArgs e)`	Responsable de vincular el evento MouseEnter del control frontal constituyente al evento MouseEnter del control de usuario.
`RJDatePicker_KeyUp(object sender, KeyEventArgs e)`	Responsable de vincular el evento KeyUp del control frontal constituyente al evento KeyUp del control de usuario.
`RJDatePicker_KeyPress(object sender, KeyPressEventArgs e)`	Responsable de vincular el evento KeyPress del control frontal constituyente al evento KeyPress del control de usuario.
`RJDatePicker_KeyDown(object sender, KeyEventArgs e)`	Responsable de vincular el evento KeyDown del control frontal constituyente al evento KeyDown del control de usuario.

Eventos

OnValueChanged	Evento por defecto del control de usuario, recreando y vinculando al evento original ValueChanged del control DateTimePicker.

8.3.4. Clase RJ TextBox

Esta clase hereda de la clase **UserControl** de la librería **Windows.Forms.**

Para crear el control RJTextBox, se agrega el control **TextBox** y **Panel** de la librería **Windows.Forms.** Implementa **propiedades de personalización de apariencia,** agregar un marcador de posición, más conocido como **placeholder,** y **expone la mayor parte de las propiedades y eventos esenciales del control TextBox** de Windows.

Propiedades

BackgroundColor	Obtiene o establece el color de fondo
BorderColor	Obtiene o establece el color de borde.
BorderSize	Obtiene o establece el tamaño de borde.
Customizable	Obtiene o establece si los colores de apariencia es personalizable.
Font	Obtiene o establece la fuente.
Multiline	Obtiene o establece un valor que indica si se trata de un control TextBox de varias líneas.
PasswordChar	Obtiene o establece el carácter utilizado para enmascarar los caracteres de una contraseña.
PlaceHolderColor	Obtiene o establece el color de texto del marcador de posición (Marca de agua).
PlaceHolderText	Obtiene o establece el texto para el marcador de posición (Marca de agua).
Style	Obtiene o establece el estilo del control (MatteBorder, FlaringBorder, MatteLine y FlaringLine)
Text	Obtiene el texto de la caja de texto.
TextColor	Obtiene o establece el color de texto de la caja de texto.

Métodos

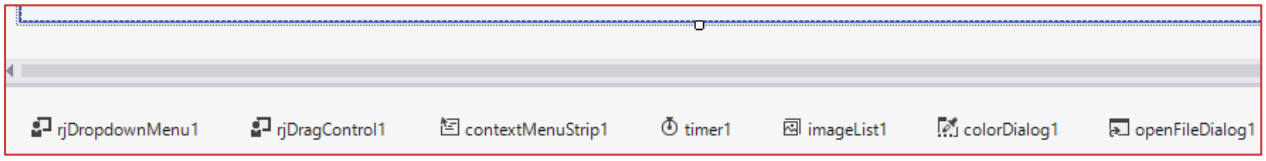
`ApplyAppearanceSettings()`	Establece la configuración de apariencia de la aplicación en las propiedades de apariencia del control.
`Clear()`	Vacía o remueve el texto de la caja de texto.
`RemovePlaceHolder()`	Quita el marcador de posición.
`RJTextBox_Click(object sender, EventArgs e)`	Responsable de vincular el evento clic del control frontal constituyente al evento clic del control de usuario.
`RJTextBox_MouseEnter( object sender, EventArgs e)`	Responsable de vincular el evento MouseEnter del control frontal constituyente al evento MouseEnter del control de usuario.
`RJTextBox_KeyUp(object sender, KeyEventArgs e)`	Responsable de vincular el evento KeyUp del control frontal constituyente al evento KeyUp del control de usuario.
`RJTextBox_KeyPress( object sender, KeyPressEventArgs e)`	Responsable de vincular el evento KeyPress del control frontal constituyente al evento KeyPress del control de usuario.
`RJTextBox_KeyDown( object sender, KeyEventArgs e)`	Responsable de vincular el evento KeyDown del control frontal constituyente al evento KeyDown del control de usuario.
`SetPlaceHolder()`	Establece el marcador de posición.
`TextBox_Enter(object sender, EventArgs e)`	Resalta el control cuando es enfocado.
`TextBox_Leave(object sender, EventArgs e)`	Desactiva el resaltado cuando el cursos deja en control.
`TextBox_TextChanged(object sender, EventArgs e)`	Responsable de vincular y ejecutar el evento por defecto recreado.

Eventos

`onTextChanged`	Evento por defecto del control de usuario, recreando y vinculando al evento original TextChanged del control TextBox.

8.4. Componentes

Los componentes **no tienen una representación visual** (A excepción de **ContextMenuStrip** ya que implementa la clase control y componente), pero es posible seleccionar, establecer propiedades y eliminarlo. Al agregar componentes al formulario mediante el diseñador **se sitúan en la parte inferior**:



Cuando instancie un componente por código **debe liberar recursos explícitamente** (Manualmente) a través de llamadas a su método **Dispose(),** o los componentes **tienen y deben implementar IContainer en su constructor** que permite asociarse con un **objeto Container**.

Por lo tanto, cuando agregue el componente mediante el diseñador (arrastrar el componente desde la caja de herramientas al formulario), **se instancia automáticamente mediante su constructor que implementa *IContainer***, por lo tanto se desechará junto con el formulario y los otros componente cuando el formulario sea cerrado. Esto es útil si usa múltiples componentes en su aplicación y desea **deshacerse de todos ellos simultáneamente y de forma segura.**

Consulte el **método Dispose(…)** de **cualquier Form.Designer.cs**, por ejemplo:

```
partial class FormComponentControls
    { //Código generado automáticamente por el diseñador.
        private System.ComponentModel.IContainer components = null;
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.dmExample = new RJDropdownMenu(this.components);
            this.dragControl = new RJDragControl(this.components);
```

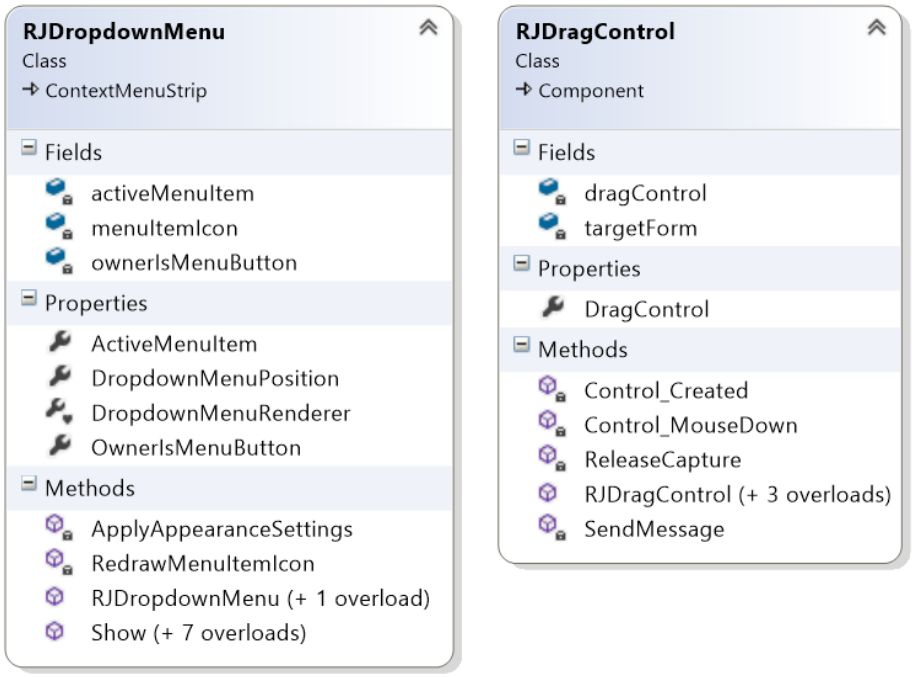
Más información:

- ✓ Componente
- ✓ ContextMenuStrip
- ✓ Timer
- ✓ Dispose - IContainer

8.4.1. Diagrama de clases

Diagrama de clases contraído

Diagrama de clases expandido



Nota:

La clase **RJDropdownMenu hereda** de la clase **ContextMenuStrip** de la librería **Windows.Forms**, y **ContextMenuStrip deriva** de la clase **Control** y la interfaz **IComponent**.

Por lo tanto ContextMenuStrip y RJDropdownMenu también **pertenecen al grupo de componentes**, **ambas clases implementan IContainer en su constructor**, no es necesario desechar los objetos manualmente si agregó al formulario mediante el diseñador, como se menciona anteriormente.

Para obtener **descripción de propiedades y métodos** del control RJDropdownMenu, **consulte el punto 8.2.4.2.**

8.4.2. Clase RJ DragControl

Esta clase hereda de la clase **Component** de la librería **System.ComponentModel.**

Este componente permite arrastrar o **mover el formulario mediante cualquier control** especificado.

Propiedades

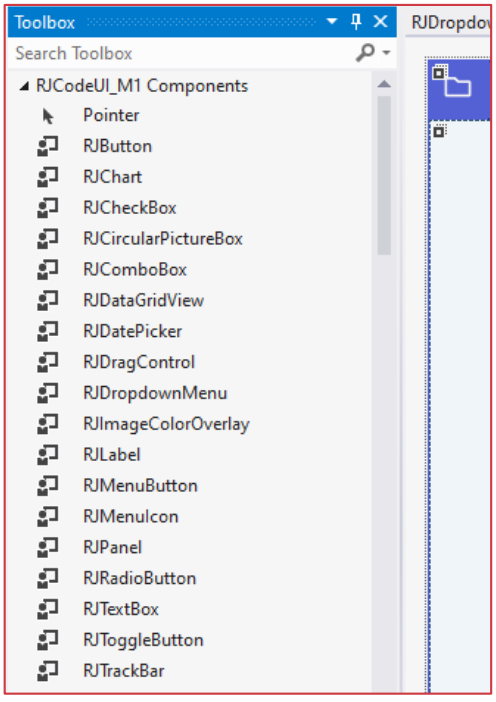
`DragControl`	Obtiene o establece el control responsable de arrastrar el formulario.

Métodos

`Control_Created(object sender, EventArgs e)`	Responsable de establecer el formulario objetivo para arrastrar.
`Control_MouseDown(object sender, MouseEventArgs e)`	Responsable de arrastrar el formulario cuando se presiona el botón del mouse.
`ReleaseCapture()`	Método externo responsable de liberar la captura del mouse en el hilo actual.
`SendMessage(IntPtr hWnd, int wMsg, int wParam, int lParam)`	Método externo responsable de enviar los mensajes a una ventana o ventanas.
`RJDragControl( IContainer container)`	Constructor con parámetro IContainer, **este constructor es invocado automáticamente en el diseñador del formulario** cuando el control es arrastrado desde la caja de herramientas al formulario. Este constructor **asegura que el objeto se elimine correctamente**.
`RJDragControl( Control _dragControl, Form _targetForm)`	Inicializa una nueva instancia con el control de arrastre y formulario especificado.
`RJDragControl( Control _dragControl, Form _targetForm, IContainer container)`	Inicializa una nueva instancia con el control de arrastre y formulario y asocia con un contenedor en especificado.

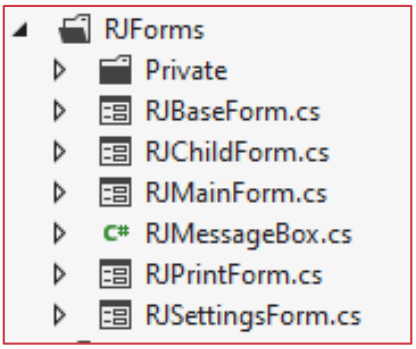
8.5.¿Cómo usarlos?

Al realizar cambios en las clases, asegúrate de **compilar el proyecto** para aplicar los cambios, los controles personalizados **se agregarán automáticamente en la caja de herramientas** de visual estudio, finalmente empieza a **arrastrarlos al formulario** como a cualquier control de Windows Form.



9. FORMULARIOS PERSONALIZADOS

Para crear formularios personalizados simplemente se **hereda un formulario existente**, luego **establecer o ampliar propiedades**, y **agregar controles** (Recomiendo hacerlo mediante código).

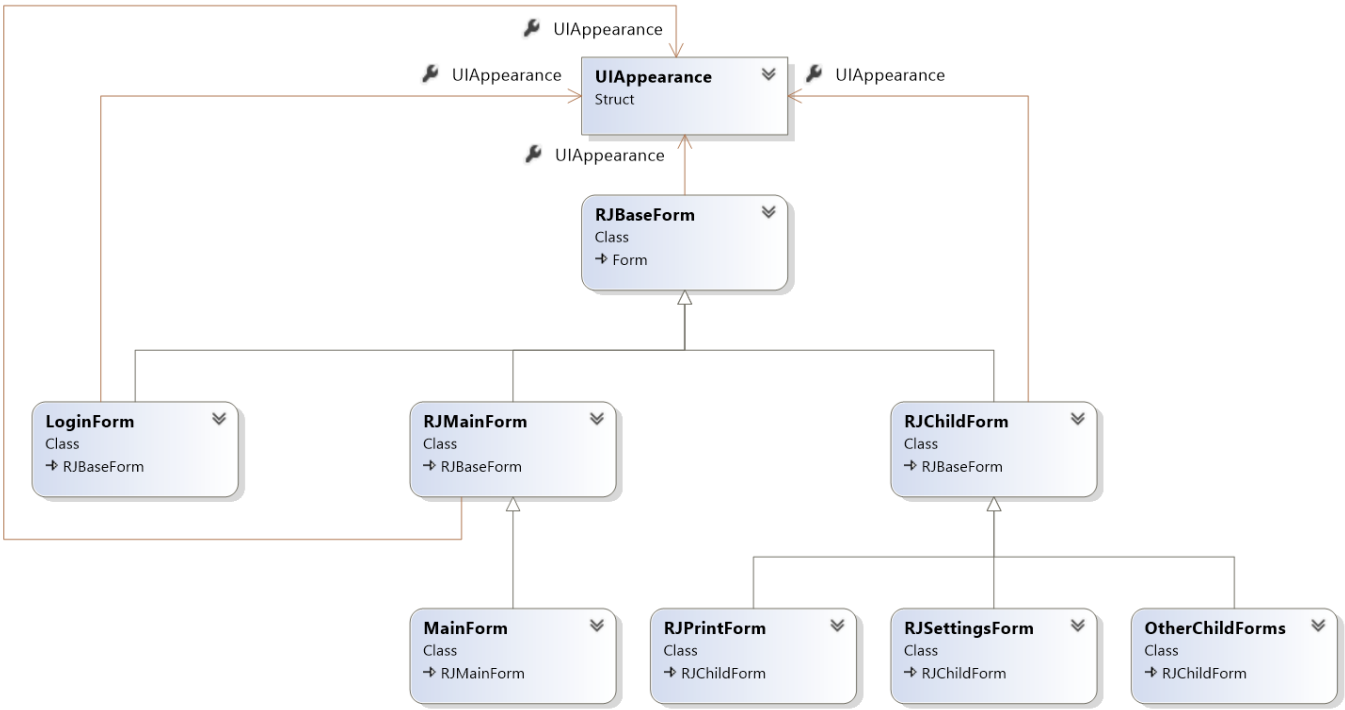


En este proyecto se crea **3 formularios personalizados base** (RJBaseForm, RJMainForm y RJChildForm) donde puedes heredar cualquier de ellos según tus necesidades y agregar tus controles.

Y se crean **4 formularios derivados** a estos formularios base, ya con un **diseño finalizado y con una función en específico** (RJSettingsForm, RJPrintForm, LoginForm y MainForm).

9.1. Diagrama de clases (Contraído)

Diagrama de clases contraído

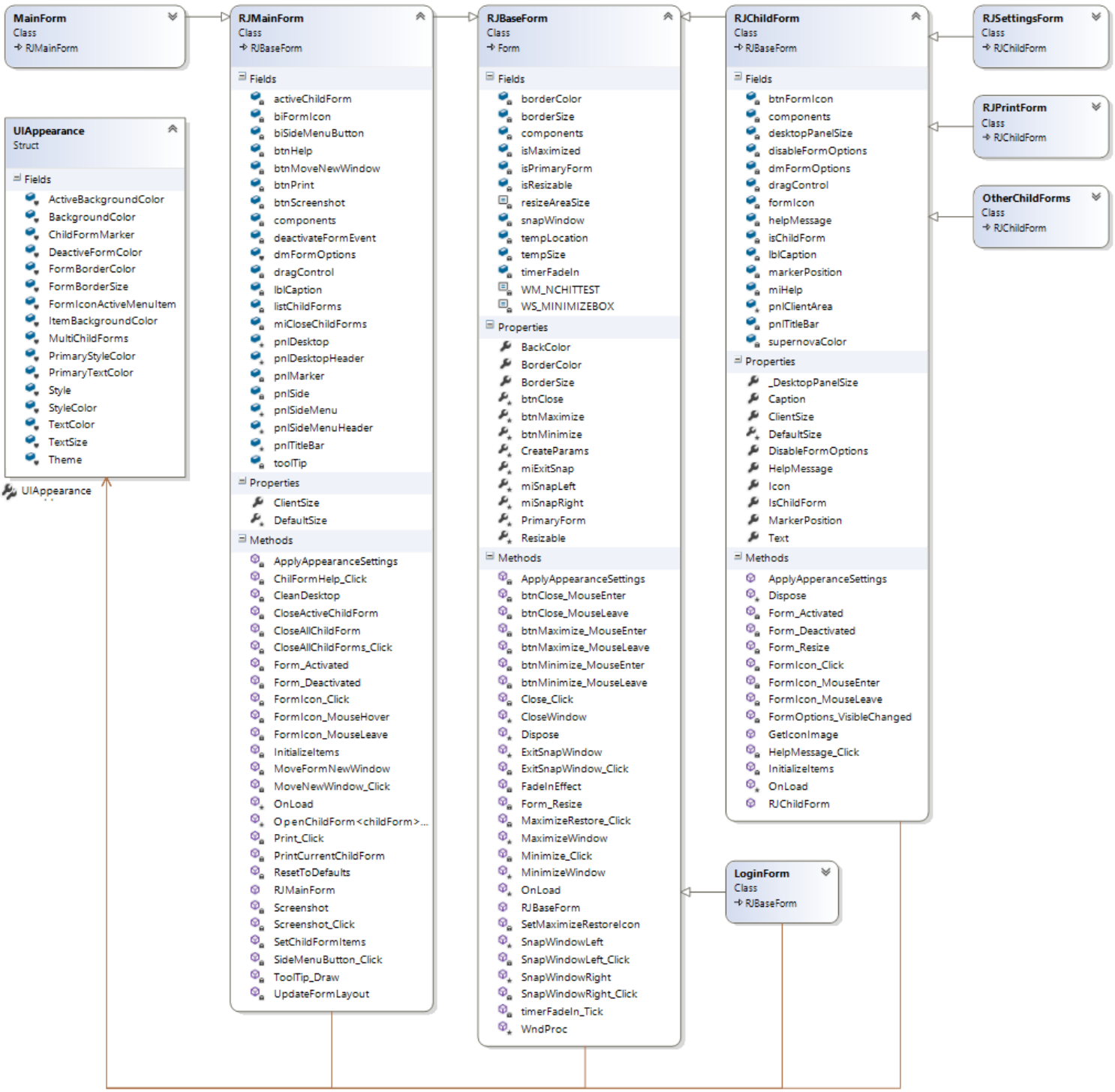


9.2. Formularios personalizados Base

Son formularios **prediseñados que puedes extender** (Heredar) para **crear rápidamente tus formularios principales o secundarios.**

RJBaseForm, RJMainForm y RJChildForm.

9.2.1. Diagrama de clases (Expandido)

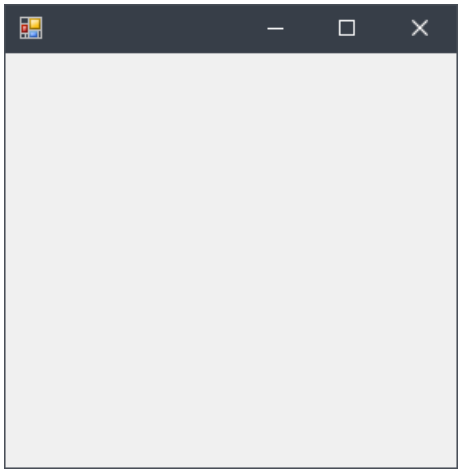


9.2.2. Formulario base - Clase RJBaseForm

Esta clase **hereda** de la clase **Form** de la librería **Windows.Forms**.

Este formulario es el **cimiento** (base) **para los demás formularios**, se encarga de **implementar todas las funcionalidades de un formulario convencional**, tales como:

- ✓ La función de **redimensionamiento**.
- ✓ La función de **maximizar, restaurar, minimizar y cerrar**.
- ✓ La función de **acoplamiento** (Snap Windows)



También se encarga de implementar algunas **propiedades básicas de apariencia**, tales como:

- ✓ Botones para la caja de control
- ✓ Tamaño de borde
- ✓ Color de borde
- ✓ Efecto Aparición gradual

Propiedades

`BorderColor`	Obtiene o establece el color de borde del formulario.
`BorderSize`	Obtiene o establece el tamaño de borde del formulario.
`btnClose`	Obtiene o establece el botón de cerrar.
`btnMaximize`	Obtiene o establece el botón maximizar.
`btnMinimize`	Obtiene o establece el botón de minimizar.
`CreateParams`	Obtiene o establece los parámetros de creación del formulario.
`miExitSnap`	Obtiene o establece el elemento de menú de salir del acoplamiento (Restaura el formulario).
`miSnapLeft`	Obtiene o establece el elemento de menú de acoplar ventana hacia la izquierda.
`miSnapRight`	Obtiene o establece el elemento de menú de acoplar ventana hacia la derecha.
`PrimaryForm`	Obtiene o establece si es formulario primario.
`Resizable`	Obtiene o establece si el formulario es **redimensionable** (cambiar de tamaño) desde los **bordes del formulario**.

Métodos

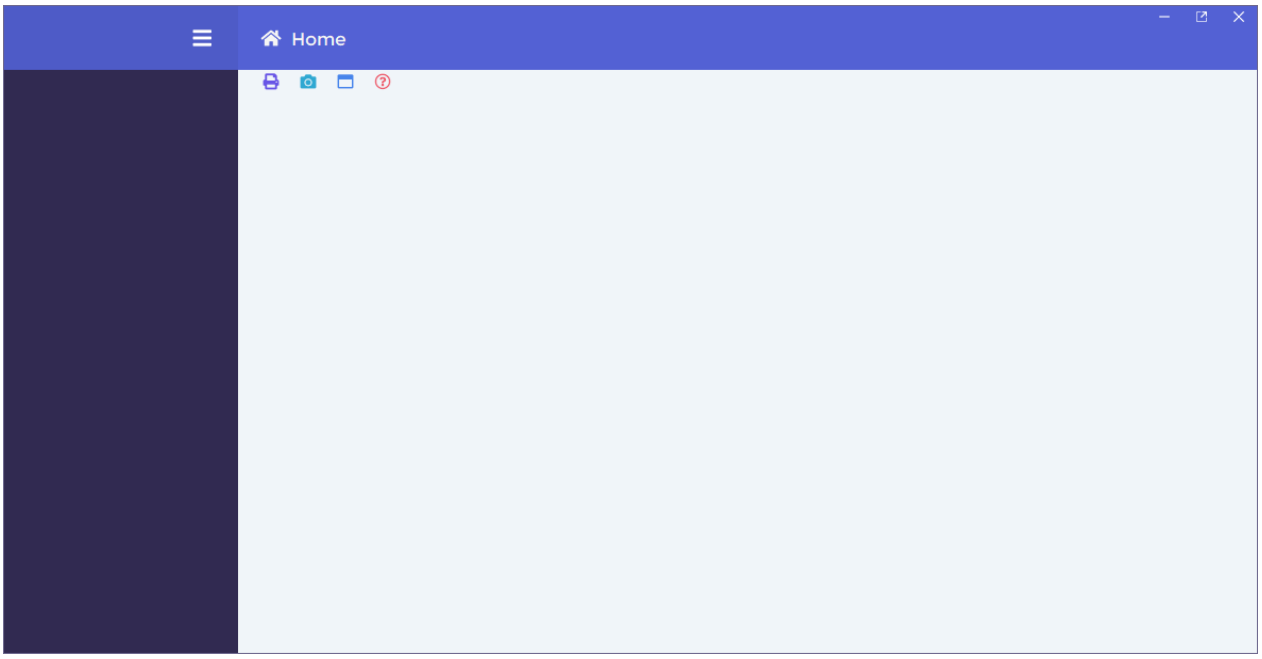
`ApplyAppearanceSettings()`	Responsable de establecer la configuración de apariencia de la aplicación a las propiedades del formulario.
`CloseWindow()`	Responsable de cerrar el formulario o salir de la aplicación si es un formulario primario.
`Dispose(bool disposing)`	Responsable de desechar el formulario y los componentes.
`ExitSnapWindow()`	Responsable de restaurar el formulario si se ha acoplado la ventana hacia uno de los lados (Snap Windows).
`FadeInEffect()`	Responsable de hacer aparecer el formulario gradualmente cuando el formulario se muestra por primera vez.
`MaximizeWindow()`	Responsable de maximizar o restaurar el formulario.
`MinimizeWindow()`	Responsable de minimizar el formulario.
`OnLoad(EventArgs e)`	Responsable de aplicar la configuración de apariencia.
`SetMaximizeRestoreIcon()`	Responsable de alternar el icono de maximizar y restaurar según sea el caso y según el tema establecido (Dark - Light).
`SnapWindowLeft()`	Responsable de acoplar el formulario hacia el lado izquierdo del escritorio.
`SnapWindowRight()`	Responsable de acoplar el formulario hacia el lado derecho del escritorio.
`WndProc(ref Message message)`	Responsable de manejar el procesamiento de mensajes de Windows a nivel del sistema operativo para el redimensionamiento del formulario.

9.2.3. Formulario principal - Clase RJMainForm

Esta clase **hereda** de la clase **RJBaseForm**.

Este es un formulario **base para el formulario principal** de su aplicación, con una **interfaz gráfica prediseñado** (Ver captura de pantalla), implementa la **barra de título** junto con los **botones de control** y una **etiqueta para mostrar el título del formulario** actualmente abierto, el **menú lateral** para agregar botones de menú junto con el icono de menú que permite contraer o expandir el menú lateral.

Finalmente implementa el **panel escritorio** para mostrar un formulario secundario en su interior y la **tira de menú** con las opciones de **captura de pantalla**, **impresión**, **ayuda** y **mover el formulario secundario a una nueva ventana.**



Propiedades

`ClientSize`	Obtiene el tamaño del área de cliente del formulario.
`DefaultSize`	Obtiene o establece el tamaño predeterminado del formulario.

Métodos

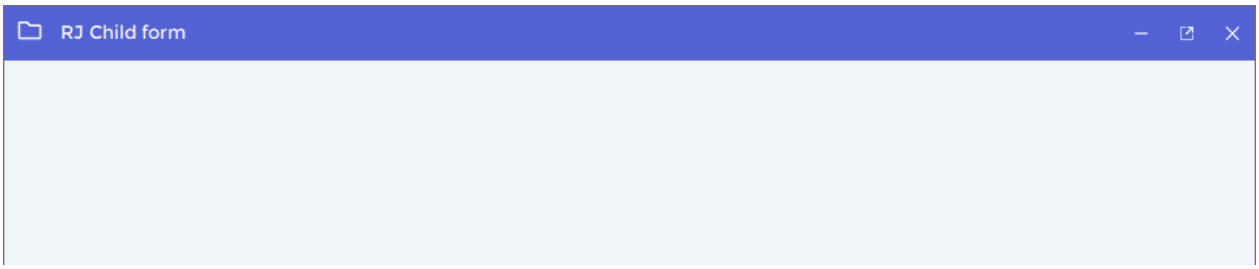
`ApplyAppearanceSettings()`	Responsable de establecer la configuración de apariencia de la aplicación a las propiedades del formulario.
`CleanDesktop()`	Responsable de limpiar el escritorio, quita el formulario secundario actual del panel escritorio.
`CloseActiveChildForm()`	Responsable de cerrar el formulario secundario actualmente abierto en el panel escritorio.
`CloseAllChildForm()`	Responsable de cerrar todos los formularios secundarios abiertos.
`InitializeItems()`	Responsable de inicializar todos los componentes del formulario principal.
`MoveFormNewWindow()`	Responsable de mover el formulario secundario actual a una nueva ventana.
`OnLoad(EventArgs e)`	Responsable de aplicar la configuración de apariencia.
`OpenChildForm<childForm>(Func<childForm> _delegate)`	Responsable de abrir un formulario secundario (RJChildForm) en el panel escritorio.

`OpenChildForm<childForm>(Func<childForm> _delegate, object senderMenuButton)`	Responsable de abrir un formulario secundario (RJChildForm) en el panel escritorio y asociarlo con un botón de menú para luego resaltarlo y establecer el marcador si es el caso.
`OpenChildForm<childForm>(Func<childForm> _delegate, object senderMenuItem, RJMenuButton ownerMenuButton)`	Responsable de abrir un formulario secundario (RJChildForm) en el panel escritorio y asociarlo con un elemento de menú y botón de menú propietario este, para luego resaltarlo y establecer el marcador si es el caso.
`PrintCurrentChildForm()`	Responsable de imprimir el formulario secundario actualmente abierto en el panel escritorio.
`ResetToDefaults()`	Responsable de restablecer los valores predeterminados del formulario.
`Screenshot()`	Responsable de tomar una captura de pantalla del formulario.
`SetChildFormItems()`	Responsable de establecer los elementos del formulario secundario en el formulario principal tales, como el icono, título y entre otros.
`UpdateFormLayout()`	Responsable de actualizar el formulario principal, se encarga de verificar si hay algún formulario secundario abierto, si es el caso vuelve a mostrarlo en el panel escritorio, caso contrario cargará sus valores predeterminados del formulario principal.

9.2.4. Formulario secundario - Clase RJChildForm

Esta clase **hereda** de la clase **RJBaseForm.**

Este es un formulario **base para los formularios secundarios** de su aplicación, con una **interfaz gráfica prediseñado** (Ver captura de pantalla), implementa la **barra de título** junto con los **botones de control**, una **etiqueta para mostrar el título del formulario** y el **menú desplegable** con las **opciones de formulario**.



Propiedades

`_DesktopPanelSize`	Obtiene o establece si el formulario debe tener el mismo tamaño del panel escritorio del formulario principal.
`Caption`	Obtiene o establece el título del formulario.

`ClientSize`	Obtiene el tamaño de área de cliente del formulario.
`DefaultSize`	Obtiene o establece el tamaño predeterminado.
`DisableFormOptions`	Obtiene o establece si se deshabilitará el menú desplegable de las opciones de formulario.
`HelpMessage`	Obtiene o establece el texto de ayuda del formulario.
`Icon`	Obtiene o establece el icono del formulario.
`IsChildForm`	Obtiene o establece si es formulario hijo del formulario principal (Hace que el formulario se muestre en el panel escritorio)
`MarkerPosition`	Obtiene o establece la posición del marcador para el botón de menú.

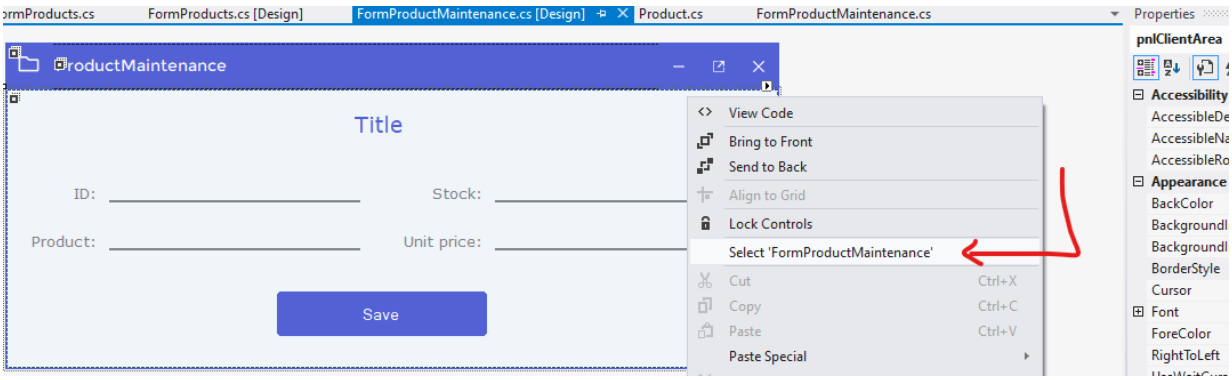
Métodos

`ApplyAppearanceSettings()`	Responsable de establecer la configuración de apariencia de la aplicación a las propiedades del formulario.
`Dispose()`	Responsable de eliminar el formulario y los componentes correctamente.
`GetIconImage (int iconSize, Color iconColor)`	Responsable de obtener el icono del formulario con tamaño y color especificados en formato de imagen.
`InitializeItems()`	Responsable de inicializar todos los componentes del formulario principal.
`OnLoad(EventArgs e)`	Responsable de aplicar la configuración de apariencia.

9.2.5. ¿Cómo usarlos?

Para usarlos **simplemente herede cualquiera de los formularios base según sus necesidades**, como se hacen a continuación en el siguiente punto.

Para **seleccionar las propiedades de un formulario**, haga **clic derecho en el diseñador** y **seleccione el formulario**, como se hace en la siguiente imagen.



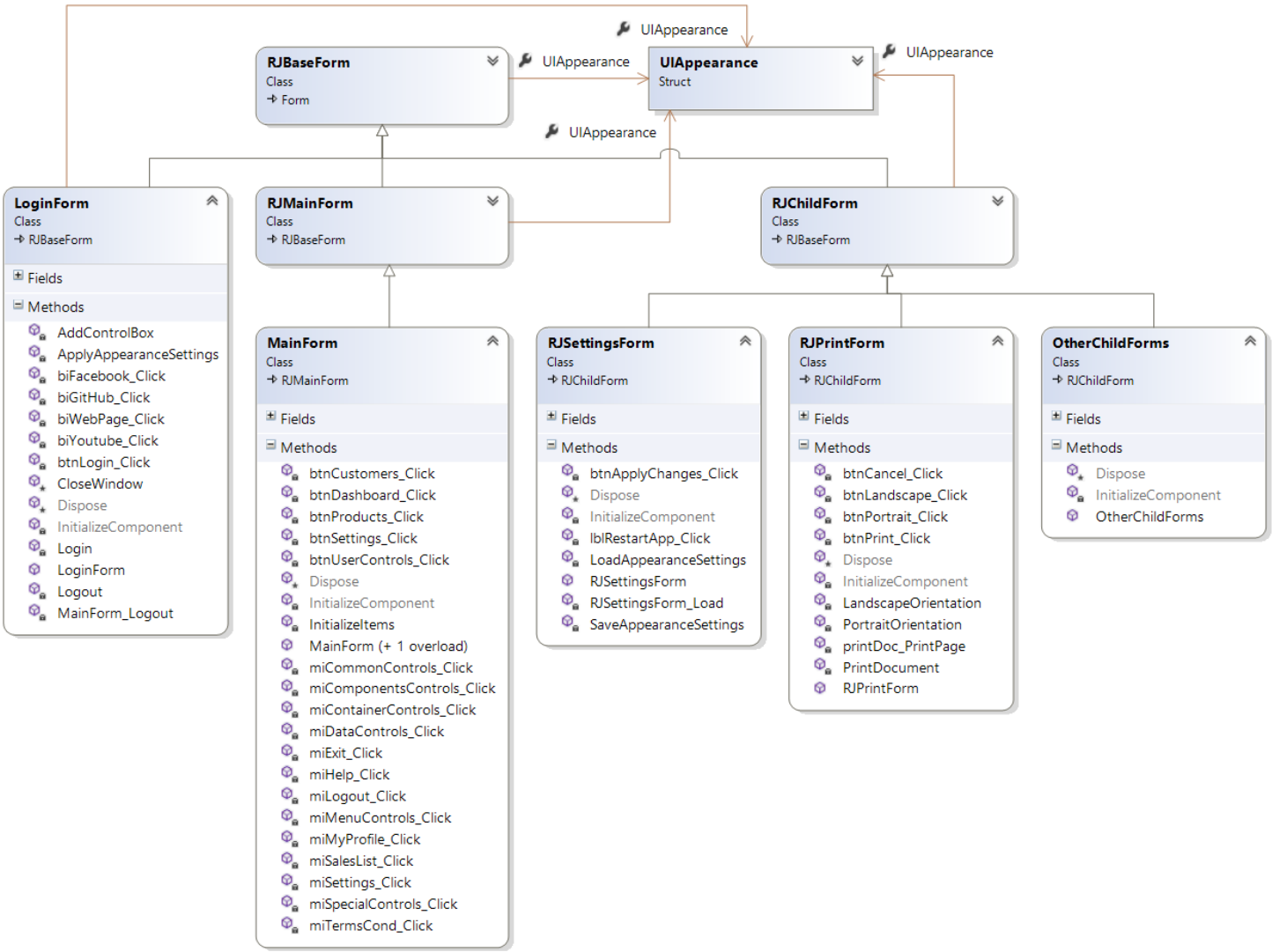
9.3. Formularios personalizados Derivados

Son formularios ya **diseñados con un propósito en específico**, por ejemplo:

- ✓ **LoginForm**: Formulario para el inicio de sesión del usuario.
- ✓ **MainForm**: Formulario principal de la aplicación.
- ✓ **RJSettingsForm**: Formulario para la configuración de apariencia de la aplicación.
- ✓ **RJPrintForm**: Formulario para imprimir una imagen en específico.

Podrías **considerarlos ejemplos de cómo utilizar los formulario personalizados base**.

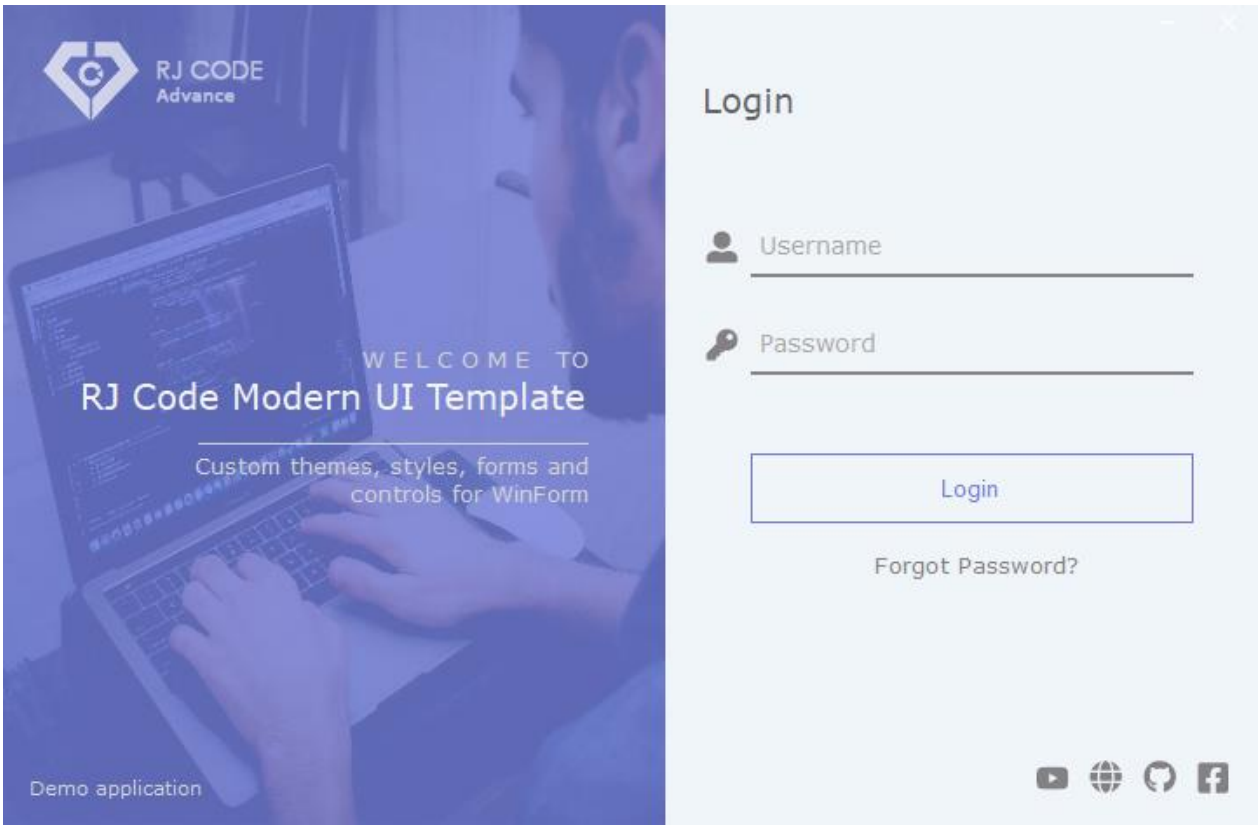
9.3.1. Diagrama de clases (Expandido)



9.3.2. Formulario de inicio de sesión - Clase LoginForm

Esta clase **hereda** de la clase **RJBaseForm**.

Formulario de **inicio de sesión del usuario,** creado mediante el diseñador.

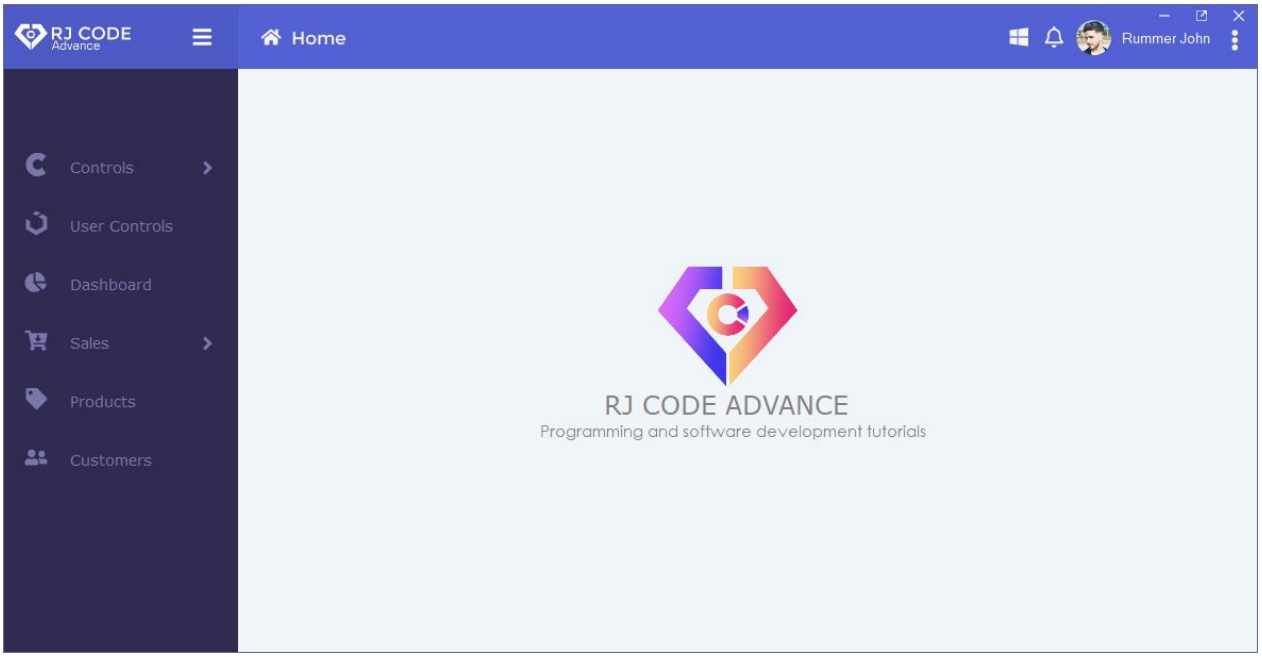


Métodos

`ApplyAppearanceSettings()`	Responsable de establecer la configuración de apariencia de la aplicación a las propiedades del formulario.
`AddControlBox()`	Responsable de eliminar el formulario y los componentes correctamente.
`Login()`	Responsable de obtener el icono del formulario con tamaño y color especificados en formato de imagen.
`Logout()`	Responsable de inicializar todos los componentes del formulario principal.

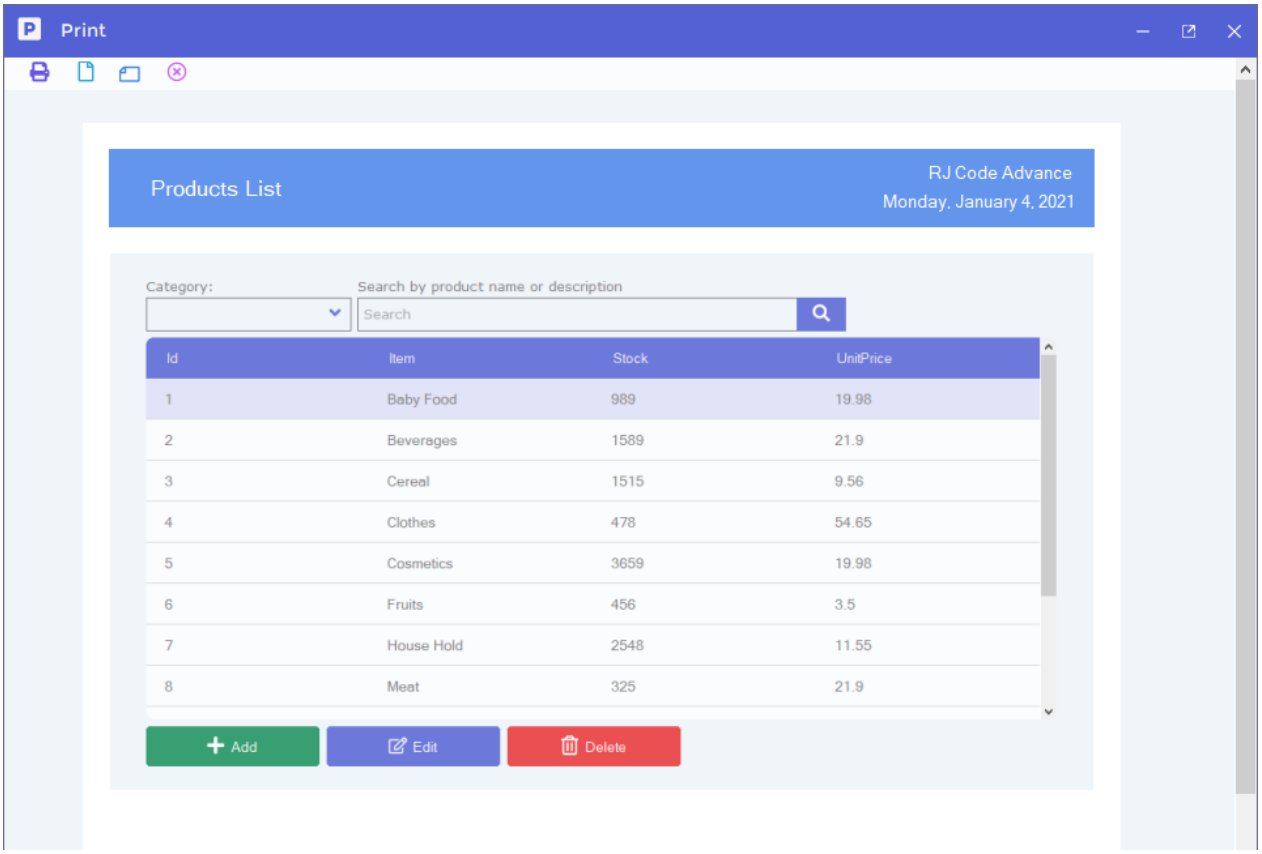
9.3.3. Formulario principal - Clase MainForm

Esta clase **hereda** de la clase **RJMainForm**.
Formulario principal de la aplicación, creado mediante el diseñador.



9.3.4. Formulario de impresión - Clase RJPrintForm

Esta clase **hereda** de la clase **RJChildForm**.
Formulario para **imprimir los formularios secundarios** del escritorio, creado mediante el diseñador.



Campos

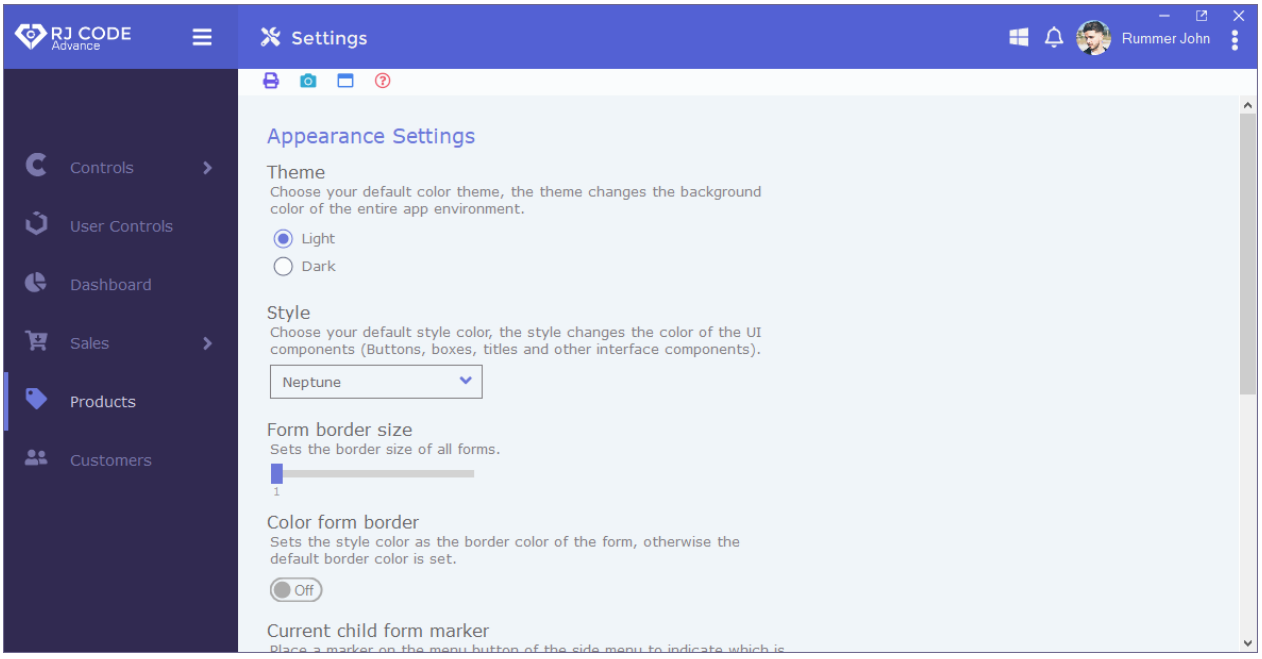
`screenshot`	Obtiene o establece la captura de pantalla a imprimir.
`sizeA4`	Obtiene o establece el tamaño del documento.
`imgDocument`	Obtiene o establece el documento en formato imagen.

Métodos

`LandscapeOrientation()`	Responsable de colocar el documento horizontalmente.
`PortraitOrientation()`	Responsable de colocar el documento verticalmente.
`PrintDocument()`	Responsable de imprimir el documento.
`RJPrintForm(Image _screenshot, string docTitle)`	Constructor con parámetros de imagen y título a imprimir.

9.3.5. Formulario de configuración - Clase RJSettingsForm

Esta clase **hereda** de la clase **RJChildForm**.
Formulario para la configuración de apariencia de la aplicación, creado mediante el diseñador.

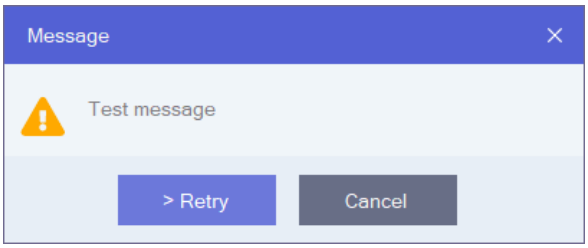


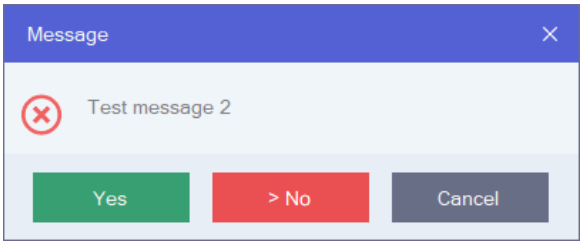
Métodos

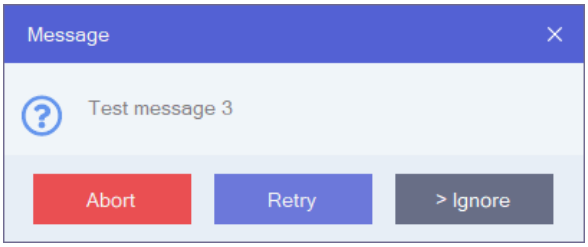
`LoadApperanceSettings()`	Se encarga de **obtener y mostrar los datos de la configuración** actual.
`SaveAppearanceSettings()`	**Guarda los cambios realizados**.

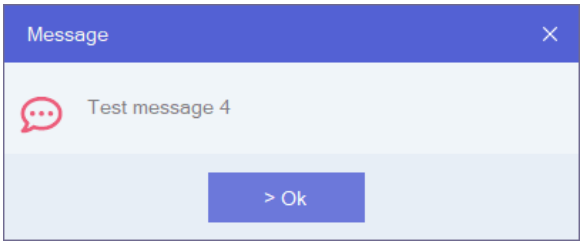
10. CUADRO DE MENSAJE PERSONALIZADO

Un cuadro de mensaje personalizado (RJMessageBox) con todas las funciones comunes del cuadro de mensaje de Windows convencional (MessageBox).










```
//Ejemplo 1
RJMessageBox.Show("Test message","Message",MessageBoxButtons.RetryCancel,
MessageBoxIcon.Exclamation);


//Ejemplo 2
RJMessageBox.Show("Test message
2","Message",MessageBoxButtons.YesNoCancel,
MessageBoxIcon.Error,MessageBoxDefaultButton.Button2);


//Ejemplo 3
RJMessageBox.Show("Test message
3","Message",MessageBoxButtons.AbortRetryIgnore,
MessageBoxIcon.Information,MessageBoxDefaultButton.Button3);


//Ejemplo 4
RJMessageBox.Show("Test message 4");
```

10.1. Diagrama de clases

Diagrama de clases contraído

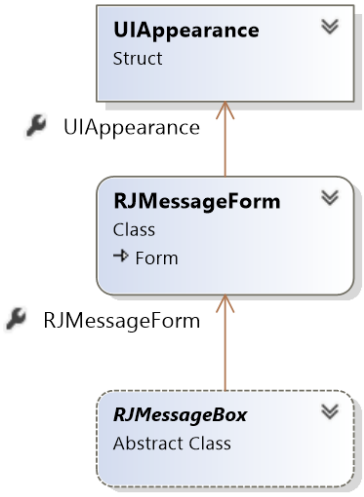
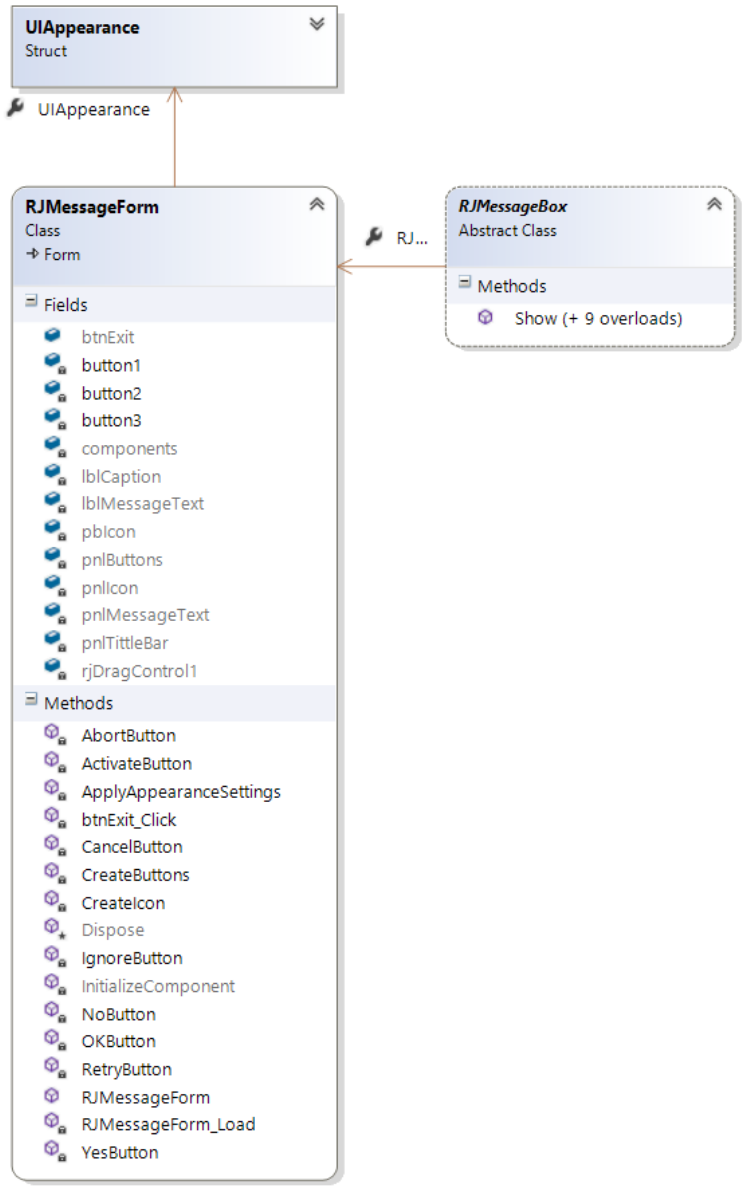


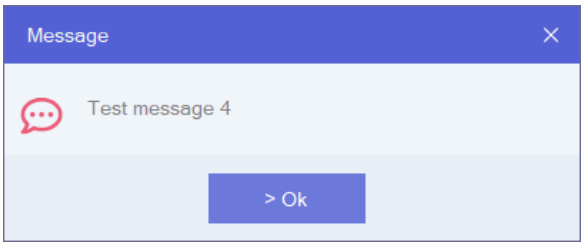
Diagrama de clases expandido



10.2. Clase RJMessageForm

Esta clase **hereda** de la clase **Form** de la librería **Windows.Forms**, fue creado mediante el diseñador.

Este formulario se encarga de la **representación visual** (Interfaz de usuario) del cuadro de mensaje.



Campos

`button1`	Botón 1 del cuadro de mensaje.
`Button2`	Botón 2 del cuadro de mensaje.
`Button3`	Botón 3 del cuadro de mensaje.

Métodos

`ApplyAppearanceSettings()`	Responsable de establecer la configuración de apariencia de la aplicación a las propiedades del formulario.
`AbortButton(int location, int locationY)`	Responsable de crear el botón de abortar.
`ActivateButton( MessageBoxDefaultButton defaultButton)`	Responsable de seleccionar (Enfocar) el botón predeterminado.
`CancelButton(int locationX, int locationY)`	Responsable de crear el botón de cancelar.
`CreateButtons( MessageBoxButtons buttons, MessageBoxDefaultButton defaultButton)`	Responsable de crear los botones especificados.
`CreateIcon( MessageBoxIcon icon)`	Responsable de crear el icono especificado.
`IgnoreButton(int locationX, int locationY)`	Responsable de crear el botón de ignorar.
`NoButton(int locationX, int locationY)`	Responsable de crear el botón de NO.

`OKButton(int locationX, int locationY)`	Responsable de crear el botón de OK.
`RetryButton(int locationX, int locationY)`	Responsable de crear el botón de reintentar.
`YesButton(int locationX, int locationY)`	Responsable de crear el botón de SI.

10.3. Clase RJMessageBox

Clase abstracta que se encargar de **mostrar el cuadro de mensaje personalizado** con los **parámetros especificados** (Ver ejemplos).

Integré la mayoría de los métodos de mostrar (*Show(…)*) del cuadro de mensaje de Windows, que tiene 21 sobrecargas de este método, y cada uno de ellos retorna un DialogResult.

Métodos

`Show(string text)`	Muestra un cuadro de mensaje con el texto especificado.
`Show(string text, string caption)`	Muestra un cuadro de mensaje con el texto y el título especificado.
`Show(string text, string caption, MessageBoxButtons buttons)`	Muestra un cuadro de mensaje con texto, título, y botones especificados
`Show(string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon)`	Muestra un cuadro de mensaje con texto, título, botones, y el icono especificado
`Show(string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon, MessageBoxDefaultButton defaultButton)`	Muestra un cuadro de mensaje con el texto, título, botones, icono y botón predeterminado especificados.
`Show(IWin32Window owner, string text)`	Muestra un cuadro de mensaje delante del objeto especificado y con el texto especificado
`Show(IWin32Window owner, string text, string caption)`	Muestra un cuadro de mensaje delante del objeto especificado y con el texto y el título especificado.
`Show(IWin32Window owner, string text, string caption, MessageBoxButtons buttons)`	Muestra un cuadro de mensaje delante del objeto especificado y con el texto, título, y botones especificados

`Show(IWin32Window owner, string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon)`	Muestra un cuadro de mensaje delante del objeto especificado y con el texto, título, botones, y el icono especificado
`Show(IWin32Window owner, string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon, MessageBoxDefaultButton defaultButton)`	Muestra un cuadro de mensaje delante del objeto especificado y con el texto, título, botones, icono y botón predeterminado especificados.

Bueno, eso es todo, espero que les haya gustado y hayan aprendido algo más de este tutorial escrito.

☺

Hasta la próxima.