# OBJETIVOS

Formalizar conceptos teóricos sobre la programación visual

# ENUNCIADO

**RESPONDA LAS SIGUIENTES PREGUNTAS**

1. **¿Qué característica posee la programación visual?**

Los lenguajes de programación visual permiten a los usuarios crear programas mediante la manipulación de elementos gráficos, en lugar de especificarlos exclusivamente de manera textual.

1. **¿Qué es un control de usuario?**

Tanto formularios como controles son clases de .Net y que pertenecen al namespaces System.Windows.Forms. A su vez, dichas clases permiten ser extendidas por el desarrollador, posibilitando ampliar las funcionalidades ya provistas por el framework.

1. **¿Cuáles son los tres elementos constitutivos que un control de usuario debe poseer?**

Propiedades, Métodos y Eventos. Lo que es cada uno (explicándoselo a alguien que conoce de programación estructurada): Las propiedades son simplemente variables. Sin embargo, a diferencia de una variable "común", éstas están estrechamente vinculadas a un objeto. Los eventos y métodos son equivalentes a los procedimientos o funciones. Y (también) están estrechamente vinculados a un objeto específico. Hay que considerar de que, en realidad, los eventos también son métodos con la única diferencia de que el evento se ejecuta automáticamente bajo ciertas condiciones.

1. **¿Para qué se utiliza la ventana denominada Solution Explorer?**

La ventana del Explorador de soluciones proporciona acceso a todos los archivos de una solución.

1. **¿Para qué se utiliza la ventana denominada Properties?**

La ventana Propiedades contiene las propiedades del formulario, control o archivo actualmente seleccionado en el IDE. Las propiedades especifican información sobre el formulario o control, como su tamaño, color y posición. Cada formulario o control tiene su propio conjunto de propiedades. Cuando selecciona una propiedad, su descripción se muestra en la parte inferior de la ventana Propiedades.

1. **¿Para qué se utiliza la ventana denominada Toolbox?**

La Caja de herramientas contiene los controles que se utilizan para personalizar los formularios. Con el desarrollo de aplicaciones visuales, se puede "arrastrar y soltar" controles en el formulario y el IDE escribirá el código que crea los controles. Esto es más rápido y sencillo que escribir (textualmente) este código (como analogía: así como no se necesita saber cómo construir un motor para conducir un automóvil, tampoco se necesita saber cómo construir controles para usarlos). La reutilización de controles preexistentes ahorra tiempo (y dinero) al desarrollar aplicaciones.

1. **¿Qué es y que puede contener una Solución en el entorno de trabajo visto en clase?**

Para comenzar el desarrollo de aplicaciones en Visual C #, se debe crear un nuevo proyecto o abrir uno existente. Un proyecto es un grupo de archivos relacionados, como el código de Visual C # y cualquier imagen que pueda formar una aplicación. Visual Studio organiza las aplicaciones en proyectos y soluciones (que contienen uno o más proyectos). Las soluciones de varios proyectos se utilizan para crear aplicaciones a gran escala.

1. **¿Qué es y que puede contener un Proyecto en el entorno de trabajo visto en clase?**

Para comenzar el desarrollo de aplicaciones en Visual C #, se debe crear un nuevo proyecto o abrir uno existente. Un proyecto es un grupo de archivos relacionados, como el código de Visual C # y cualquier imagen que pueda formar una aplicación. Visual Studio organiza las aplicaciones en proyectos y soluciones (que contienen uno o más proyectos). Las soluciones de varios proyectos se utilizan para crear aplicaciones a gran escala.

1. **¿Qué es un formulario?**

Cada formulario es un objeto de la clase Form en la biblioteca de clases de NET Framework.

1. **Describa al menos 10 propiedades, 10 métodos y 10 eventos del control Button.**

|  |  |  |
| --- | --- | --- |
| **PROPERTIES** | **METHODS** | **EVENTS** |
| AccessibilityObject | AccessibilityNotifyClients(AccessibleEvents, Int32) | AutoSizeChanged |
| AccessibleDefaultActionDescription | AccessibilityNotifyClients(AccessibleEvents, Int32, Int32) | BackColorChanged |
| AccessibleDescription | BeginInvoke(Delegate) | BackgroundImageChanged |
| AccessibleName | BeginInvoke(Delegate, Object[]) | BackgroundImageLayoutChanged |
| AccessibleRole | BringToFront() | BindingContextChanged |
| AllowDrop | Contains(Control) | CausesValidationChanged |
| Anchor | CreateAccessibilityInstance() | ChangeUICues |
| AutoEllipsis | CreateControl() | Click |
| AutoScrollOffset | CreateControlsInstance() | ClientSizeChanged |
| AutoSize | CreateGraphics() | ContextMenuChanged |

1. **Describa al menos 10 propiedades, 10 métodos y 10 eventos del control CheckBox.**

|  |  |  |
| --- | --- | --- |
| **PROPERTIES** | **METHODS** | **EVENTS** |
| AccessibilityObject | AccessibilityNotifyClients(AccessibleEvents, Int32) | AppearanceChanged |
| AccessibleDefaultActionDescription | AccessibilityNotifyClients(AccessibleEvents, Int32, Int32) | AutoSizeChanged |
| AccessibleDescription | BeginInvoke(Delegate) | BackColorChanged |
| AccessibleName | BeginInvoke(Delegate, Object[]) | BackgroundImageChanged |
| AccessibleRole | BringToFront() | BackgroundImageLayoutChanged |
| AllowDrop | Contains(Control) | BindingContextChanged |
| Anchor | CreateAccessibilityInstance() | CausesValidationChanged |
| Appearance | CreateControl() | ChangeUICues |
| AutoCheck | CreateControlsInstance() | CheckedChanged |
| AutoEllipsis | CreateGraphics() | CheckStateChanged |

1. **Describa al menos 10 propiedades, 10 métodos y 10 eventos del control Combobox.**

Las siguientes propiedades se aplican a la mayoría de los objetos:

1) **NAME:** Establece el nombre del control, a través del cual se puede acceder a las propiedades y métodos del control.  
2) **APPEARANCE:** puede ser 0 para un aspecto plano o 1 para un aspecto tridimensional.  
3) **BACK COLOR:** establece el color de fondo en el que se muestra el texto o se dibujan los gráficos.  
4) **FORE COLOR:** establece el color de primer plano  
5) **FONT:** establece la cara, el atributo y el tamaño de la fuente utilizada para el texto en el control.  
6) **TITLE:** establece el texto que se muestra en muchos controles que no aceptan entrada, por ejemplo, el texto en un control de etiqueta, el título de un control de botón de comando.  
7) **TEXT:** establece el texto que se muestra en los controles que aceptan la entrada del usuario, por ejemplo, el control TextBox.  
8) **WIDTH/HEIGHT:** estas propiedades establecen las dimensiones del control.  
9) **LEFT/TOP:** estas propiedades establecen las coordenadas de la esquina superior izquierda del control, expresadas en las unidades del contenedor.  
10) **ENABLED:** de forma predeterminada, el valor de esta propiedad es Verdadero, lo que significa que el control puede obtener el foco.  
11) **VISIBLE:** establezca esta propiedad en False para que un control sea invisible.

1. **Describa al menos 10 propiedades, 10 métodos y 10 eventos del control DateTimePicker.**

Las siguientes propiedades se aplican a la mayoría de los objetos:

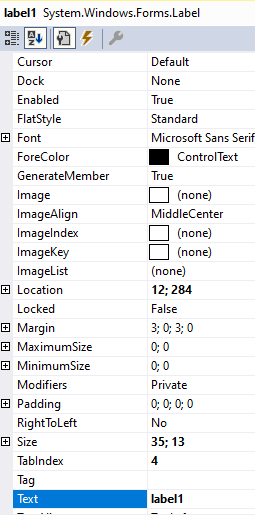
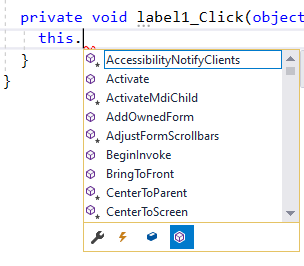
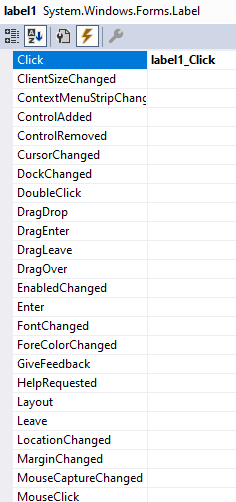
1) **NAME:** Establece el nombre del control, a través del cual se puede acceder a las propiedades y métodos del control.  
2) **APPEARANCE:** puede ser 0 para un aspecto plano o 1 para un aspecto tridimensional.  
3) **BACK COLOR:** establece el color de fondo en el que se muestra el texto o se dibujan los gráficos.  
4) **FORE COLOR:** establece el color de primer plano  
5) **FONT:** establece la cara, el atributo y el tamaño de la fuente utilizada para el texto en el control.  
6) **TITLE:** establece el texto que se muestra en muchos controles que no aceptan entrada, por ejemplo, el texto en un control de etiqueta, el título de un control de botón de comando.  
7) **TEXT:** establece el texto que se muestra en los controles que aceptan la entrada del usuario, por ejemplo, el control TextBox.  
8) **WIDTH/HEIGHT:** estas propiedades establecen las dimensiones del control.  
9) **LEFT/TOP:** estas propiedades establecen las coordenadas de la esquina superior izquierda del control, expresadas en las unidades del contenedor.  
10) **ENABLED:** de forma predeterminada, el valor de esta propiedad es Verdadero, lo que significa que el control puede obtener el foco.  
11) **VISIBLE:** establezca esta propiedad en False para que un control sea invisible.

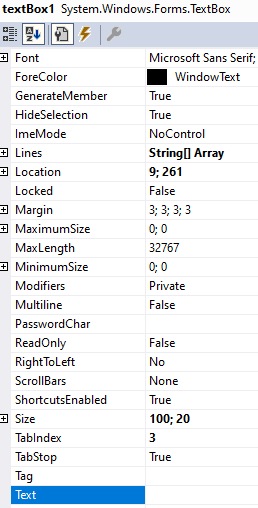
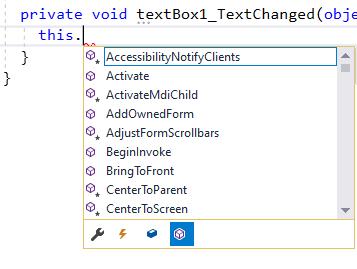
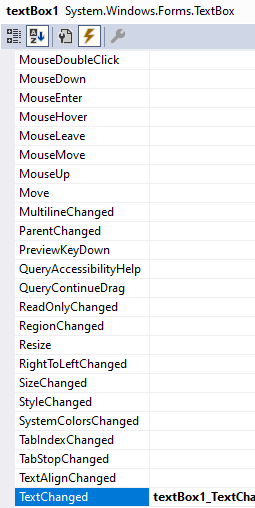
1. **Describa al menos 10 propiedades, 10 métodos y 10 eventos del control Timer.**

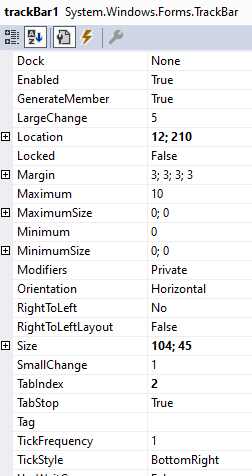
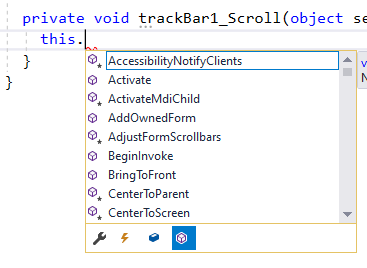
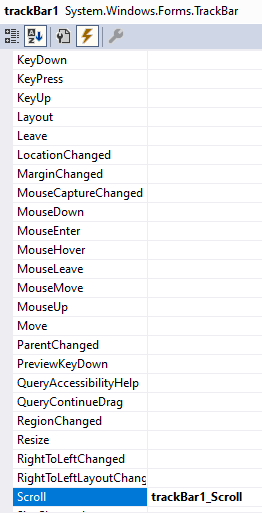
Las siguientes propiedades se aplican a la mayoría de los objetos:

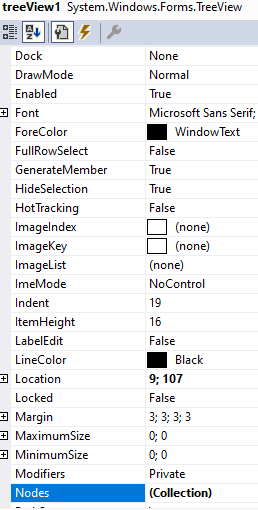
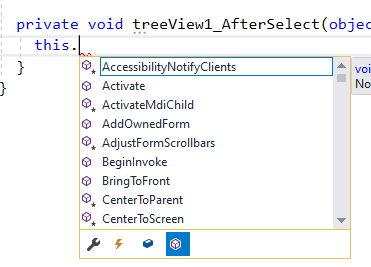
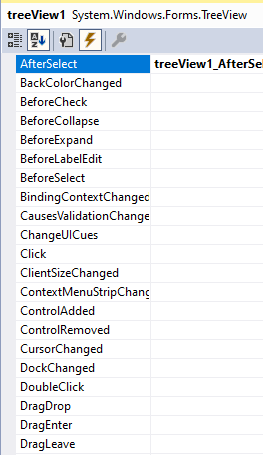
1) **NAME:** Establece el nombre del control, a través del cual se puede acceder a las propiedades y métodos del control.  
2) **APPEARANCE:** puede ser 0 para un aspecto plano o 1 para un aspecto tridimensional.  
3) **BACK COLOR:** establece el color de fondo en el que se muestra el texto o se dibujan los gráficos.  
4) **FORE COLOR:** establece el color de primer plano  
5) **FONT:** establece la cara, el atributo y el tamaño de la fuente utilizada para el texto en el control.  
6) **TITLE:** establece el texto que se muestra en muchos controles que no aceptan entrada, por ejemplo, el texto en un control de etiqueta, el título de un control de botón de comando.  
7) **TEXT:** establece el texto que se muestra en los controles que aceptan la entrada del usuario, por ejemplo, el control TextBox.  
8) **WIDTH/HEIGHT:** estas propiedades establecen las dimensiones del control.  
9) **LEFT/TOP:** estas propiedades establecen las coordenadas de la esquina superior izquierda del control, expresadas en las unidades del contenedor.  
10) **ENABLED:** de forma predeterminada, el valor de esta propiedad es Verdadero, lo que significa que el control puede obtener el foco.  
11) **VISIBLE:** establezca esta propiedad en False para que un control sea invisible.

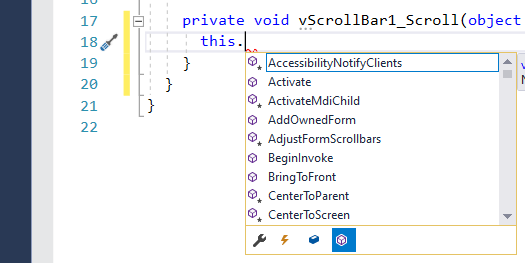
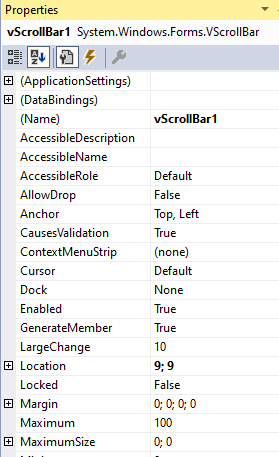
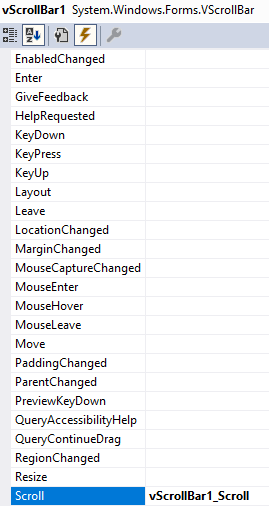
1. **Seleccione 5 controles adicionales y de cada uno de ellos describa 10 métodos, 10 propiedades y 10 eventos.**

**LABEL:** 

**TEXTBOX:** 

**TRACKBAR:** 

**TREEVIEW:** 

**VSCROLLBAR:** 

# Parte 2

1. **¿Cómo se declara un procedimiento? Ejemplo.**

Declaración de clase:

class Bienvenido {

// Cuerpo del procedimiento.

}

1. **¿Cómo se declara una función? Ejemplo**

**FORMATO:**

<Access Specifier> <Return Type> <Method Name>(Parameter list)

Body

}

**EJEMPLO:**

public void add()

{

Body

}

1. **¿Qué diferencia existe entre una función y un procedimiento?**

La función devuelve un valor, el procedimiento, no.

1. **¿Qué significa que el parámetro de un procedimiento se pasa por valor?**

Cuando un argumento se pasa por valor (el valor predeterminado en C #), se realiza una copia de su valor y se pasa al método llamado. Los cambios en la copia no afectan el valor de la variable original en el llamador. Esto evita los efectos secundarios accidentales que dificultan enormemente el desarrollo de sistemas de software correctos y fiables.

Cuando se pasa un argumento por referencia, el llamador le da al método la capacidad de acceder y modificar la variable original del llamador; no se pasa ninguna copia.

1. **¿Qué significa que el parámetro de una función se pasa por referencia?**

Cuando un argumento se pasa por valor (el valor predeterminado en C #), se realiza una copia de su valor y se pasa al método llamado. Los cambios en la copia no afectan el valor de la variable original en el llamador. Esto evita los efectos secundarios accidentales que dificultan enormemente el desarrollo de sistemas de software correctos y fiables.

Cuando se pasa un argumento por referencia, el llamador le da al método la capacidad de acceder y modificar la variable original del llamador; no se pasa ninguna copia.

# Parte 3

1. **Enumere las estructuras de decisión. Ejemplifique cada una de ellas y explique en qué se diferencian.**

**if:** le indicamos una condición; si esta es verdadera, se ejecuta:

if (condicion) {

// Instrucciones

}

**Switch:** esta estructura condicional es de selección múltiple: le damos un valor (puede ser una variable) y una lista de casos, y si cumple alguno de los casos, ejecuta las instrucciones asociadas a ella:

switch (valor){

case caso1:

// Instrucciones

break;

case caso2:

// Instrucciones

break;

case caso N:

// Instrucciones

break;

default:

// Instrucciones

break;

}

1. **Enumere las estructuras de repetición. Ejemplifique cada una de ellas y explique en qué se diferencian.**

int counter = 0;

while (counter < 10)

{

Console.WriteLine($"Hello World! The counter is {counter}");

counter++;

}

La instrucción while comprueba una condición y ejecuta la instrucción o el bloque de instrucciones que aparece después de while. La comprobación de la condición y la ejecución de dichas instrucciones se repetirán hasta que la condición sea false.

int counter = 0;

do

{

Console.WriteLine($"Hello World! The counter is {counter}");

counter++;

} while (counter < 10);

Este bucle do y el bucle while anterior generan el mismo resultado.

for (int index = 0; index < 10; index++)

{

Console.WriteLine($"Hello World! The index is {index}");

}

El código anterior funciona de la misma forma que los bucles while y do que ya ha usado.

**Diferencias:**

**For:** Ejectuará el bucle la cantidad de especificada, no importa qué.

**Do:** Ejectuará al menos una vez las sentencias, y luego evaluará si continúa.

**While:** No ejecutará nada hasta que evalúe primero la condición dada al bucle.

1. **¿Cómo se crea una estructura en el entorno visto en clase?**

En el editor de código.

1. **¿Qué es una propiedad?**

Las propiedades son simplemente variables. Sin embargo, a diferencia de una variable "común", éstas están estrechamente vinculadas a un objeto.

1. **¿Qué es un método?**

Los métodos son equivalentes a los procedimientos o funciones vistos en programación estructurada (la diferencia es que están vinculados a un objeto específico).

1. **¿Qué es un evento?**

Los eventos son equivalentes a los procedimientos o funciones vistos en programación estructurada (la diferencia es que están vinculados a un objeto específico y que son de ejecución automática dadas ciertas condiciones).

1. **¿Cuál es el uso ideal de una variable de tipo Boolean?**

En variables que solo pueden sostener por su naturaleza los valores TRUE y FALSE.

1. **¿Para que se usa el método Parse cuando se lo utiliza por ejemplo como Decimal.Parse?**

Los PARSE son procedimiento para convertir una cadena en un número.

1. **¿Cómo se declara un vector de 10 posiciones de tipo Decimal?**

float vector[10];

1. **¿Cuándo hablamos de boxing y unboxing a que nos referimos?**

El boxing (empaquetado) y unboxing (desempaquetado) en C # permite a los desarrolladores convertir tipos de datos NET de tipo de valor a tipo de referencia y viceversa. La conversión de un tipo de valor en un tipo de referencia se denomina boxing en C # y la conversión de un tipo de referencia en un tipo de valor se denomina unboxing.