OBJETIVOS

Formalizar conceptos teóricos sobre la programación visual

ENUNCIADO

Responda las siguientes preguntas

1. ¿Qué característica posee la programación visual?

Las características de la programación visual son:

Permite que el usuario pueda interactuar con la aplicación de una forma más sencilla en comparación con las aplicaciones de consola.

Facilita al programador al momento de utilizar los controles ya que estos ya están definidos con sus propiedades y métodos para ser utilizados.

2. ¿Qué es un control de usuario?

Un control de usuario es un elemento gráfico que permite interactuar con el usuario, este control de usuario puede ser un botón, un cuadro de texto, una etiqueta que describe al cuadro de texto, grillas para realizar una Alta Baja o Modificación de datos (ABM), etc.

En C# .NET, los controles de usuario son proporcionados por WindowsForms y permite programar su comportamiento en base a las acciones que haga el usuario. De los controles anteriores mencionados su equivalente serían; button, textBox, label y dataGridView. Todos los controles son clases en el que se pueden utilizar sus métodos para crear la funcionalidad.

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

Los tres elementos constitutivos que un control de usuario debe poseer son las propiedades, métodos y los eventos.

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

La ventana Solution Explorer se utiliza para ver todos los proyectos que están incluidos en una solución. Dicha ventana se puede utilizar para crear nuevos proyectos, nuevos elementos (como formularios o clases), seleccionar un proyecto para compilarlo, etc.

5. ¿Para qué se utiliza la ventana denominada Properties?

La ventana Properties sirve para ver todas las propiedades que tiene asociado un control en específico. En esta ventana se pueden modificar propiedades como Name, Text, BackColor, Cursor, Padding, Margin, Opacity, Size, etc.

Resulta importante usar la propiedad Name porque es el nombre del control que se utilizará después para programarlo en los métodos.

6. ¿Para qué se utiliza la ventana denominada Toolbox?

La ventana ToolBox permite visualizar todos los controles que hay en WindowsForms, cada control tiene sus propiedades y eventos asociados. En esta ventana se puede filtrar para buscar un control en específico o se puede navegar entre las diferentes categorías de controles como Controles Comunes, Contenedores, Componentes, General, etc.

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

Una solución es un contenedor de varios proyectos relacionados y de una carpeta llamada .vs que guarda archivos de compilación. Se puede modificar los proyectos de la solución en la ventana Solution Explorer de Visual Studio.

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

Un proyecto contiene tres elementos:

Dependencias: dentro de Dependencias se encuentran los Analizadores y Marcos de trabajo. Para instalar dependencias se hace uso del gestor de paquetes NuGet visible en Solution Explorer.

Form1.cs: si se creó un proyecto usando WindowsForms, se va a crear este archivo que es un formulario, se puede acceder al formulario mediante el diseñador o el código. Se pueden agregar más formularios si se desea.

Program.cs: es el punto de entrada donde se va a ejecutar la aplicación, dentro de Main se hace un Application.Run y se hace la instanciación del formulario principal.

9. ¿Qué es un formulario?

Un formulario es una ventana o cuadro de diálogo que constituye la interfaz de usuario de una aplicación. Al igual que el control, posee Propiedades, Métodos y Eventos. Además, el formulario sirve de contenedor para los demás controles de usuario.

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

Propiedades:

1. (Name): indica el nombre utilizado en el código para identificar al objeto.
2. Anchor: define a qué bordes del contenedor está enlazado un control.
3. AutoSize: especifica si un control cambiará automáticamente de tamaño para ajustarse a su contenido.
4. BackColor: el color del fondo del componente.
5. BackgroundImage: imagen de fondo utilizado para el control.
6. Cursor: cursor que aparece al pasar el puntero por el control.
7. Size: tamaño del control, en píxeles.
8. Text: texto asociado al control.
9. TextAlign: alineación del texto que se mostrará en el control.
10. Visible: determina si el control está visible u oculto.

Métodos:

1. Select(): activa el control.
2. Update(): hace que el control vuelva a dibujar las regiones no válidas en su área de cliente.
3. Hide(): oculta el control de usuario.
4. Focus(): establece el foco de entrada en el control.
5. Contains(Control): obtiene un valor que indica si el control especificado es un control secundario del control.
6. Equals(Object): determina si el objeto especificado es igual al objeto actual.

7. GetType(): obtiene el Type de la instancia actual.
8. Refresh(): obliga al control a invalidar su área cliente y acto seguido, obliga a que vuelva a dibujarse el control y sus controles secundarios.
9. Show(): muestra el control al usuario.
10. ToString(): devuelve un String que contiene el nombre del Component, si existe. Este método no se debe invalidar.

Eventos:

1. Click: se produce cuando se hace un click en el control.
2. ClientSizeChanged: se produce cuando cambia el valor de la propiedad ClientSize.
3. DoubleClick: se produce cuando el usuario hace doble click en el control Button.
4. DragDrop: se produce cuando se completa una operación de arrastrar y colocar.
5. DragEnter: se produce cuando se arrastra un objeto dentro de los límites del control.
6. Enter: se produce cuando se entra al control.
7. FontChanged: se produce cuando cambia el valor de la propiedad Font.
8. GotFocus: se produce cuando el control recibe el foco.
9. KeyUp: se produce cuando se suelta una tecla mienras el control tiene el foco.
10. Move: se produce cuando se mueve el control.

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

Propiedades:

1. AccessibleDescription: descripción que se proporciona a clientes con discapacidades para facilitar el acceso.
2. (DataBindings): enlaces de datos para el control.
3. AllowDrop: indica si el control puede aceptar datos que el usuario arrastre.
4. Appearance: controla la apariencia de la casilla.
5. AutoCheck: hace que la casilla cambie automáticamente de estado cuando se selecciona.
6. CausesValidation: indica si este componente genera eventos de validación.
7. CheckAlign: determina la ubicación de la casilla dentro del control.
8. Checked: indica si el componente se encuentra en el estado de activación.
9. CheckState: indica el estado del componente.
10. ThreeState: indica si CheckBox permitirá tres estados de activación en lugar de dos.

Métodos:

1. BeginInvoke(Delegate): ejecuta el delegado especificado de forma asincrónica en el subproceso donde se creó el identificador subyacente del control.
2. BringToFront(): coloca el control al principio del orden Z.
3. CreateControl(): fuerza la creación del control visible, incluidos el identificador y los controles secundarios visibles.
4. FindForm(): recupera el formulario en el que se encuentra el control.
5. Hide(): oculta el control al usuario.
6. InitLayout(): se llama a este método cuando el control se ha agregado a otro contenedor.
7. IsInputChar(Char): determina si un carácter es un carácter de entrada que el control reconoce.

8. IsInputKey(Keys): determina si la tecla especificada es una tecla de entrada normal o una tecla especial que requiere preprocesamiento.
9. OnClick(EventArgs): genera el evento click.
10. OnControlAdded(ControlEventArgs): genera el evento ControlAdded.

Eventos:

1. AppearanceChanged: se produce cuando cambiar el valor de la propiedad Appearance.
2. AutoSizeChanged: se produce cuando cambia el valor de la propiedad AutoSize.
3. BackColorChanged: se produce cuando cambia el valor de la propiedad BackColor.
4. DataContextChanged: se produce cuando cambia el valor de la propiedad DataContext.
5. DragDrop: se produce cuando se completa una operación de arrastrar y colocar.
6. DragLeave: se produce cuando se arrastra un objeto fuera de los límites del control.
7. DragOver: se produce cuando se arrastra un objeto sobre los límites del control.
8. EnabledChanged: se produce cuando cambia el valor la propiedad Enabled.
9. GiveFeedback: se produce durante una operación de arrastre.
10. Invalidated: se produce cuando es necesario volver a dibujar un control.

12. Describa al menos 10 propiedades, 10 métodos y 10 eventos del control ComboBox.

Propiedades:

1. AccessibleName: nombre que se proporciona a clientes con discapacidades para facilitar el acceso.
2. AccessibeRole: rol que se proporciona a clientes con discapacidades para facilitar el acceso.
3. AutoCompleteMode: indica el comportamiento de finalización del texto del cuadro combinado.
4. AutoCompleteSource: el origen de cadenas completas utilizadas para la finalización automática.
5. CausesValidation: indica si este componente genera eventos de validación.
6. Items: elementos en el cuadro combinado.
7. Sorted: indica si están ordenados los elementos en la lista del cuadro combinado.
8. TabIndex: indica el índice del orden de tabulación que ocupará este control.
9. MaxLength: especifica el número máximo de caracteres que pueden introducirse en un cuadro combinado.
10. DisplayMember: indica la propiedad que se va a mostrar de los elementos de este control.

Métodos:

1. CreateHandle(): crea un identificador para el control.
2. DestroyHandle(): destruye el identificador asociado a este control.
3. Dispose(): libera todos los recursos que usa ComboBox.
4. Dispose(Boolean): libera todos los recursos no administrados que usa ComboBox y, de forma opcional, libera los recursos administrados.
5. DoDragDrop(Object, DragDropEffects): inicia una operación de arrastrar y colocar.

6. DoDragDrop(Object, DragDropEffects, Bitmap, Point, Boolean): comienza una operación de arrastre.
7. BeginUpdate(): mantiene el rendimiento cuando se agregan elementos a ComboBox de uno en uno.
8. EndUpdate(): reanuda el dibujo del control ComboBox después de que el método BeginUpdate() suspenda el dibujo.
9. BringToFront(): coloca el control al principio del orden Z.
10. FindString(String, Int32): devuelve el índice del primer elemento de ComboBox después del índice especificado que contiene la cadena especificada. La búsqueda no hace distinción de mayúsculas y minúsculas.

Eventos:

1. DrawItem: se produce cuando cambia la apariencia de un control ComboBox dibujado por el propietario.
2. DropDown: aparece cuando se muestra la parte desplegable de un ComboBox.
3. DropDownClosed: se produce cuando la parte desplegable del elemento ComboBox ya no está visible.
4. DropDownStyleChanged: se produce cuando ha cambiado la propiedad DropDownStyle.
5. TabIndexChanged: se produce cuando cambia el valor de la propiedad TabStop.
6. TextChanged: se produce cuando cambia el valor de la propiedad Text.
7. TextUpdate: se produce cuando el control ha dado formato al texto, pero antes de que este se muestre.
8. Validated: se mproduce cuando finaliza la validación del control.
9. Validating: se produce cuando el control se está validando.
10. ValueMemberChanged: se produce cuando cambia la propiedad ValueMember.

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

Propiedades:

1. CalendarFont: fuente utilizada para mostrar el calendario.
2. CalendarForeColor: color utilizado para mostrar texto en un mes.
3. CalendarMonthBackground: color de fondo que se muestra dentro del mes.
4. CalendarTitleBackColor: color de fondo que se muestra en el título del calendario.
5. CalendarTitleForeColor: color utilizado para mostrar texto en el título del calendario.
6. CalendarTrailingForeColor: color utilizado para mostrar el texto de los días anteriores y siguientes que complementan el mes. Los días anteriores y siguientes son aquellos de los meses anterior y posterior que aparecen en el calendario del mes actual.
7. CustomFormat: cadena personalizada de formato utilizada para dar formato a la fecha y hora que se muestran en el control.
8. Format: determina si el formato de fechas y hora se muestra con formato estándar o personalizado.
9. MaxDate: fecha máxima que se puede seleccionar.
10. MinDate: fecha mínima que se puede seleccionar.

Métodos:

1. ToString(): devuelve una cadena que representa el control DateTimePicker actual.
2. ResetBackColor(): restablece el valor predeterminado de la propiedad BackColor.
3. ResetBindings(): hace que un control enlazado a BindingSource vuelva a leer todos los elementos de la lista y actualice los valores mostrados.
4. ResetCursor(): restablece el valor predeterminado de la propiedad Cursor.
5. ResetFont(): restablece el valor predeterminado de la propiedad Font.
6. ResetForeColor(): restablece el valor predeterminado de la propiedad ForeColor.
7. ResetText(): restablece el valor predeterminado de la propiedad Text.
8. OnHandleCreated(EventArgs): genera el evento HandleCreated.
9. OnHandleDestroyed(EventArgs): genera el evento HandleDestroyed.
10. OnHelpRequested(HelpEventArgs): genera el evento HelpRequested.

Eventos:

1. ValueChanged: tiene lugar cuando el valor del control cambia.
2. VisibleChanged: tiene lugar cuando cambia la visibilidad del control.
3. CloseUp: tiene lugar cuando el usuario ha terminado de seleccionar una fecha del calendario desplegable.
4. CursorChanged: evento que se desencadena cuando se cambia el val or de la propiedad Cursor de Control.
5. FormatChanged: evento que se desencadena cuando se cambia el valor de la propiedad Format de DateTimePicker.
6. HelpRequested: tiene lugar cuando el usuario solicita ayuda para el control.
7. Leave: tiene lugar cuando el control ya no es el control activo del formulario.
8. LocationChanged: evento que se desencadena cuando se cambia el valor de la propiedad Location de Control.
9. RegionChanged: evento que se desencadena cuando el valor de la propiedad Region cambia de Control.
10. SystemColorsChanged: evento que se desencadena cuando cambian los colores del sistema.

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

Propiedades:

1. (Name): indica el nombre utilizado en el código para identificar al objeto.
2. Enabled: habilita la generación de eventos Elapsed.
3. GenerateMember: indica si se generará una variable miembro para este componente.
4. Interval: frecuencia de los eventos Elapsed en milisegundos.
5. Modifiers: indica el nivel de visibilidad del objeto.
6. Tag: datos definidos por el usuario asociados con el objeto.

Métodos:

1. Start(): inicia el temporizador.
2. Stop(): detiene el temporizador.
3. ToString(): devuelve una cadena que representa Timer.
4. OnTick(EventArgs): genera el evento Tick.
5. MemberwiseClone(): crea una copia superficial del Object actual.
6. InitializeLifetimeService(): obtiene un objeto de servicio de duración para controlar la directiva de duración de esta instancia.

7. GetType(): obtiene el Type de la instancia actual.
8. GetService(Type): devuelve un objeto que representa el servicio suministrado por Compoment o por Container.
9. GetHashCode(): sirve como la función hash predeterminada.
10. CreateObjRef(Type): crea un objeto que contiene toda la información relevante necesaria para generar un proxy utilizado para comunicarse con un objeto remoto.

Eventos:

1. Tick: tiene lugar cuando ha transcurrido el intervalo de tiempo especificado.
2. Disposed: tiene lugar cuando una llamada elimina el componente mediante una llamada al método Dispose().

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

GroupBox

Propiedades:

1. TabStop: obtiene o establece un valor que indica si el usuario puede presionar la tecla TAB para dar el foco al control GroupBox.
2. MaximumSize: obtiene o establece el tamaño que es el límite superior que GetPreferredSize(Size) puede especificar.
3. MinimumSize: obtiene o establece el tamaño que es el límite inferior que GetPreferredSize(Size) puede especificar.
4. Font: obtiene o establece la fuente del texto que muestra el control.
5. Handle: obtiene el identificar de ventana al que está enlazado el control.

Métodos:

1. CreateGraphics(): crea el Graphics para el control.
2. CreateHandle(): crea un identificar para el control.
3. GetAccessibilityObjectById(Int32): recupera el AccessibleObject especificado.
4. GetChildAtPoint(Point): recupera el control secundario ubicado en las coordenadas especificadas.
5. ProcessKeyMessage(Message): procesa un mensaje de teclado.

Eventos:

1. Enter: tiene lugar cuando el control se convierte en el control activo del formulario.
2. FontChanged: evento que se desencadena cuando se cambia el valor de la propiedad Font.
3. LocationChanged: evento que se desencadena cuando se cambia el valor de la propiedad Location.
4. MarginChanged: tiene lugar cuando la propiedad Margin ha cambiado.
5. MouseHover: tiene lugar cuando el mouse permanece quieto dentro del control durante un tiempo.

Label

Propiedades:

1. Dock: define los bordes del control enlazados al contenedor.
2. Enabled: indica si el control está habilitado.
3. FlatStyle: determina la apariencia del control cuando un usuario mueve el mouse sobre el control y hace click.
4. GenerateMember: indica si se generará una variable miembro para este componente.
5. Size: tamaño del control, en píxeles.

Métodos:

1. Select(): activa el control.
2. Update(): hace que el control vuelva a dibujar las regiones no válidas en su área de cliente.
3. Focus(): establece el foco de entrada en el control.
4. GetAutoSizeMode(): recupera un valor que indica cómo se comportará un control cuando su propiedad AutoSize esté habilitada.
5. GetContainerControl(): devuelve el siguiente ContainerControl (en sentido ascendente) de la cadena de controles primarios del control.

Eventos:

1. BackColorChanged: evento que se desencadena cuando se cambia el valor de la propiedad BackColor de Control.
2. ControlRemoved: tiene lugar cuando un control se quite de este control.
3. CursorChanged: evento que se desencadena cuando se cambia el valor de la propiedad Cursor de Control.
4. DoubleClick: tiene lugar cuando se hace doble click en el componente.
5. MouseDown: tiene lugar cuando el puntero del mouse se encuentra sobre el componente y se pulsa un botón del mouse.

LinkLabel

Propiedades:

1. ActiveLinkColor: determina el color del hipervínculo cuando el usuario hace click en el vínculo.
2. AllowDrop: indica si el control puede aceptar datos que el usuario arrastre.
3. DisabledLinkColor: determina el color del hipervínculo cuando está deshabilitado.
4. LinkVisited: determina si el hipervínculo debe presentarse como visitado.
5. VisitedLinkColor: determina el color del hipervínculo cuando la propiedad LinkVisited está establecida en true.

Métodos:

1. GetNextControl(Control, Boolean): Recupera el siguiente control, hacia delante o hacia atrás, en el orden de tabulación de controles secundarios.
2. OnMouseEnter(EventArgs): genera el evento MouseEnter.
3. OnMouseHover(EventArgs): genera el evento MouseHover.
4. OnMouseLeave(EventArgs): genera el evento OnMouseLeave(EventArgs).
5. OnMove(EventArgs): genera el evento Move.

Eventos:

1. LinkClicked: tiene lugar cuando se hace click en el vínculo.
2. MouseClick: tiene lugar cuando se hace click con el mouse en el control.
3. MouseDoubleClick: tiene lugar cuando se hace doble click con el mouse en el control.
4. LocationChanged: evento que se desencadena cuando se cambia el valor de la propiedad Location de Control.
5. Layout: tiene lugar cuando el control va a disponer su contenido.

ListBox

Propiedades:

1. DataSource: indica la lista que este control utiliza para obtener sus elementos.
2. DisplayMember: indica la propiedad que se va a mostrar de los elementos de este control.
3. FormatString: caracteres de especificador de formato que indican como se muestra un valor.
4. Items: elementos en el cuadro de lista.
5. MultiColumn: indica si los valores deben mostrarse horizontalmente por columnas.

Métodos:

1. ClearSelected(): anula la selección de todos los elementos del control ListBox.
2. BeginUpdate(): mantiene el rendimiento mientras se agregan elementos del control ListBox de uno en uno al impedir que se dibuje el control hasta que se llame el método EndUpdate().
3. FindString(String): busca el primer elemento del control ListBox que comience por la cadena especificada.
4. FindStringExact(String): busca el primer elemento del control que coincida exactamente con la cadena especificada.
5. FindStringExact(String, Int32): busca el primer elemento del control ListBox que coincida exactamente con la cadena especificada. La búsqueda comienza con un índice de inicio específico.

Eventos:

1. SelectIndexChanged: tiene lugar cuando el valor de la propiedad SelectIndex cambia.
2. SelectValueChanged: evento que se desencadena cuando se cambia el valor de la propiedad SelectedValue de ListControl.
3. FormatStringChanged: evento que se desencadena cuando el valor de la propiedad FormatString cambia.
4. ImeModeChanged: tiene lugar cuando cambia el modo IME (Editor de métodos de entrada) del control.
5. MeasureItem: tiene lugar cuando se necesita calcular el alto de algún elemento determinado.

MenuStrip

Propiedades:

1. ContextMenuStrip: menú contextual que se muestra cuando el usuario hace click con el botón secundaria en el control.
2. Enabled: indica si el control está habilitado.
3. Items: colección de elementos que se van a mostrar en ToolStrip.
4. LayoutStyle: especifica la orientación del diseño de ToolStrip.
5. ShowItemToolTips: especifica si se van a mostrar informaciones sobre herramientas en los elementos.

Métodos:

1. AdjustFormScrollbars(Boolean): ajusta las barras de desplazamiento del contenedor en función de las posiciones actuales de los controles y del control que está seleccionado.
2. CreateControlIsInstance(): crea una nueva instancia de la colección de controles para el control.
3. CreateDefaultItem(String, Image, EventHandler): crea un elemento de menú ToolStripMenuItem con el texto, imagen y controlador de eventos especificados en un nuevo control MenuStrip.
4. CreateLayoutSettings(ToolStripLayoutStyle): especifica la organización visual para el elemento ToolStrip.
5. Dispose(Boolean): libera los recursos no administrados que usa ToolStrip y, de forma opcional, libera todos los recursos administrados.

Eventos:

1. ItemClicked: tiene lugar cuando se hace click en el elemento.
2. KeyPress: tiene lugar cuando el control tiene foco y el usuario presiona y suelta una tecla.
3. MenuActivate: tiene lugar cuando el usuario ha comenzado a tener acceso al menú a través del teclado o mouse.
4. MenuDeactivate: tiene lugar cuando el usuario ha terminado de obtener acceso al menú mediante el teclado o el mouse.
5. Scroll: tiene lugar cuando el usuario mueve el cuadro de desplazamiento.

HelpProvider

Propiedades:

1. (Name): indica el nombre utilizado en el código para identificar el objeto.
2. GenerateMember: indica si se generará una variable miembro para este componente.
3. HelpNamespace: espacio de nombres de ayuda (por ejemplo, archivo de Ayuda HTML 1.0) que se utilizará. Sólo se utiliza en controles en los que se ha establecido HelpKeyword.
4. Modifiers: indiva el nivel de visibilidad del objeto.
5. Tag: datos definidos por el usuario asociados con el objeto.

Métodos:

1. ToString(): devuelve una cadena que representa el objeto HelpProvider actual.
2. GetType(): obtiene el Type de la instancia actual.

3. GetShowHelp(Control): devuelve un valor que indica si se debe mostrar la ayuda del control especificado.
4. ResetShowHelp(Control): quita la cadena de ayuda asociada al control especificado.
5. SetHelpString(Control, String): especifica la cadena de ayuda asociada al control especificado.

Eventos:

1. Disposed: tiene lugar cuando una llamada elimina el componente mediante una llamada al método Dispose().

ErrorProvider

Propiedades:

1. BlinkRate: velocidad en milisegundos con la que parpadea el icono de error.
2. BlinkStyle: controla si el icono de error parpadea al establecer un error.
3. ContainerControl: control principal, generalmente el formulario, que contiene los controles enlazados a datos en los que ErrorProvider puede mostrar iconos de error.
4. Icon: icono utilizado para indicar un error.
5. DataMember: indica la sublista de datos de DataSource en los que se deben enlazar errores.

Métodos:

1. Clear(): borra todos los valores de configuración asociados a este componente.
2. GetError(Control): devuelve la cadena de descripción del error actual para el control especificado.
3. GetIconAlignment(Control): obtiene un valor que indica dónde se debe colocar el icono de error con respecto al control.
4. GetIconPadding(Control): devuelve el espacio adicional que se va a dejar junto al icono de error.
5. SetError(Control, String): establece la cadena de descripción del error para el control especificado.

Eventos:

1. RightToLeftChanged: tiene lugar cuando el valor de la propiedad RightToLeft cambia.
2. Disposed: tiene lugar cuando una llamada elimina el componente mediante una llamada al método Dispose().

FileSystemWatcher

Propiedades:

1. EnableRaisingEvents: obtiene o establece un valor que indica si el componente está habilitado.
2. Filter: obtiene o establece la cadena de filtro utilizada para determinar qué archivos se supervisan en un directorio (.txt, .jpg, *).
3. IncludeSubdirectories: obtiene o establece un valor que indica si se deben supervisar los subdirectorios de la ruta de acceso especificada.
4. Path: obtiene o establece la ruta de acceso del directorio que se va a inspeccionar.

5. SynchronizingObject: obtiene o establece el objeto utilizado para serializar las llamadas del controlador de eventos emitidas como consecuencia de un cambio de directorio.

Métodos:

1. OnChanged(FileSystemEventArgs): genera el evento Changed.
2. OnCreated(FileSystemEventArgs): genera el evento Created.
3. OnDeleted(FileSystemEventArgs): genera el evento Deleted.
4. OnError(FileSystemEventArgs): genera el evento Error.
5. OnRenamed(FileSystemEventArgs): genera el evento Renamed.

Eventos:

1. Changed: se produce cuando cambia un archivo o un directorio en la ruta de acceso Path especificada.
2. Created: se produce cuando se crea un archivo o un directorio en la ruta de acceso Path especificada.
3. Deleted: se produce cuando se elimina un archivo o un directorio en la ruta de acceso Path especificada.
4. Error: se produce cuando la instancia de FileSystemWatcher no puede continuar supervisando los cambios o cuando el búfer interno se desborda.
5. Renamed: se produce cuando se cambia de nombre un archivo o un directorio en la ruta de acceso Path especificada.

PropertyGrid

Propiedades:

1. CanShowVisualStyleGlyphs: establece si se pueden usar los glifos visuales habilitados para el sistema operativo en los nodos de expansión del área de la cuadrícula.
2. CategoryForColor: color del texto utilizado para los encabezados de categorías. El color de fondo lo determina la propiedad LineColor.
3. CategorySplitterColor: color de la línea que separa las categorías.
4. HelpVisible: establece si se va a mostrar o no el panel de descripciones.
5. PropertySort: establecerá el tipo de ordenación que PropertyGrid utilizará para mostrar propiedades.

Métodos:

1. CollapseAllGridItems(): contrae todas las categorías de PropertyGrid.
2. ExpandAllGridItems(): expande todas las categorías de PropertyGrid.
3. OnPropertySortChanged(EventArgs): genera el evento PropertySortChanged.
4. UpdateStyles(): obliga a que los estilos asignados vuelvan a aplicarse al control.
5. Update(): hace que el control vuelva a dibujar las regiones no válidas en su área de cliente.

Eventos:

1. Click: tiene lugar cuando se hace click en el componente.
2. ChangeUICues: tiene lugar cuando se muestran u ocultan los rectángulos de foco y los subrayados de señal del teclado.
3. PropertySortChanged: tiene lugar cuando la propiedad PropertySort de PropertyGrid ha cambiado.

4. RightToLeftChanged: tiene lugar cuando el valor de la propiedad RightToLeft cambia.
5. SelectedGridItemChanged: tiene lugar cuando la cuadricula seleccionada ha cambiado.

SplitContainer

Propiedades:

1. Dock: define los bordes del control enlazados al contenedor.
2. ContextMenuStrip: menú contextual que se muestra cuando el usuario hace click con el botón secundario en el control.
3. SplitterIncrement: determina el número de píxeles que se mueve el divisor en incrementos. De manera predeterminada es 1.
4. SplitterDistance: determina la distancia en píxeles del divisor desde el borde izquierdo o superior.
5. IsSplitterFixed: determina si el divisor se puede mover.

Métodos:

1. GetScrollState(Int32): determina si se ha establecido el marcador especificado.
2. GetTopLevel(): determina si el control es de nivel superior.
3. Refresh(): obliga al control a invalidar su área cliente y, acto seguido, obliga a que vuelva a dibujarse el control y sus controles secundarios.
4. ScaleControl(SizeF, BoundsSpecified): escala la ubicación, el tamaño, el relleno y el margen.
5. SetClientSizeCore(Int32, Int32): establece el tamaño del área cliente del control.

Eventos:

1. Panel1: panel izquierdo o superior del SplitContainer.
2. Panel2: panel derecho o inferior del SplitContainer.
3. SizeChanged: evento que se desencadena cuando se cambia el valor de la propiedad Size de Control.
4. SplitterMoving: tiene lugar cuando se mueve el divisor.
5. SplitterMoved: tiene lugar cuando se ha terminado de mover el divisor.

16. ¿Cómo se declara un procedimiento? Ejemplo.

Para declarar un procedimiento primero se indica su visibilidad (privado, público o protegido) y su tipo de retorno que es void, el nombre de la función y los parámetros tipados. La palabra reservada void indica que no devuelve nada y es lo que caracteriza a un procedimiento.

Ejemplo:

```
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    monthCalendar1.MinDate = DateTime.Now;
}
```

17. ¿Cómo se declara una función? Ejemplo.

Para declarar una función primero se indica su visibilidad, el tipo de dato que va a retornar, nombre de la función y los parámetros tipados. Al final de la función se tiene que hacer un return de la variable, respetando el tipo de dato con el cuál se declaró a la función.

Ejemplo:

```
1 referencia
static double ConversorKBaBits(double pesoKB)
{
    return (pesoKB * Math.Pow(2, 10)) * 8;
}
```

Nota: en este ejemplo, "*static"* significa que no es necesario instanciar la clase para usar la función.

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

La diferencia entre una función y un procedimiento es que la función retorna un valor y el procedimiento no retorna nada. En los códigos de ejemplo, la función retorna un *double* y el procedimiento un *void*.

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

Cuando se pasa un parámetro por valor, se hace una copia del valor original y se asigna a una nueva variable dentro del procedimiento. Esto significa que los cambios realizados en el parámetro dentro del procedimiento no afectan a la variable original fuera de él.

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

Cuando se pasa un parámetro por referencia en una función, se trabajará con la variable original ya que se estará tomando como parámetro la dirección de memoria de dicha variable. Cualquier modificación que se haga en la función afectará a la variable original. En C# la sintaxis para declarar una referencia se usa la palabra reservada "*ref*".

21. Enumere las estructuras de decisión. Ejemplifique cada una de ellas y explique en que se diferencian.

1. Estructura de decisión **IF**:

```
if(textBox1.Text.Length == 0)
{
    MessageBox.Show("Ingrese un numero", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

En este tipo de IF, si la expresión es true, ejecuta el código que está entre llaves.

2. Estructura de decisión **IF-ELSE**:

```
if (textBox6.Text == "ABC")
{
    textBox6.Text = "Si";
}
else
{
    textBox6.Text = "No";
}
```

El IF-ELSE ejecuta código cuando la expresión es true y también ejecuta código cuando la expresión es false.

3. Estructura de decisión **ELSE-IF**:

```
if (a == 1) //Si a es igual a 1
{
    Console.WriteLine("a vale 1");
}
else if (a == 2) //Si a es igual a 2
{
    Console.WriteLine("a vale 2");
}
else //Si no se cumple ninguna condición anterior
{
    Console.WriteLine("a no vale ni 1 ni 2")
}
```

En los ELSE-IF se evalúan múltiples condiciones y si no coincide con ninguna, ejecuta el código que está en ELSE.

4. Estructura de decisión **Switch**:

```
int? val = null;
string output = null;
switch (val)
{
    case 1:
        output = "A";
        break;
    case 2:
        output = "B";
        break;
    case 3:
        output = "C";
        break;
    default:
        output = "default";
        break;
}
```

La estructura Switch es parecida a ELSE-IF, con la diferencia de que Switch usa como condición valores específicos como números o letras, a diferencia de los IF que pueden usar operadores lógicos.

5. Estructura de decisión **ternaria**:

```
int edad = 23;
var comprobacionEdad = edad > 18 ? "Mayor de edad" : "Menor de edad";
Console.Write($" El usuario es: {comprobacionEdad}");
```

Es una forma corta de declarar un IF-ELSE. Si la condición es true, ejecuta lo de la derecha de “?”, si es false, ejecuta lo de la derecha de “:”.

22. Enumere las estructuras de repetición. Ejemplifique cada una de ellas y explique en que se diferencian.

1. Estructura de repetición **WHILE**:

```
int n = 0;
while (n < 5)
{
    Console.Write(n);
    n++;
}
```

En la sentencia While primero se evalúa la condición, si es true, ejecuta el código. Para detener el bucle se necesita incrementar n, de esta forma habrá 5 iteraciones.

2. Estructura de repetición **DO-WHILE**:

```
int n = 0;
do
{
    Console.Write(n);
    n++;
} while (n < 5);
```

En la sentencia Do-While primero se ejecuta el código y después se evalúa la condición. Igual que el ejemplo anterior, la forma de parar el bucle es incrementando n.

3. Estructura de repetición FOR:

```
for (int i = 0; i < 3; i++)
{
    Console.Write(i);
}
```

La sentencia For es útil cuando ya se sabe cuántas iteraciones tendrá el bucle, se usa típicamente para recorrer Arrays y hacer operaciones con sus elementos. La estructura del For consiste en inicializar el iterador, hasta donde debe llegar y su incremento.

4. Estructura de repetición Foreach:

```
List<int> fibNumbers = new() { 0, 1, 1, 2, 3, 5, 8, 13 };
foreach (int element in fibNumbers)
{
    Console.Write($"{element} ");
}
```

Adicionalmente, la sentencia Foreach permite iterar cada elemento de una colección pudiendo ejecutar un bloque de código para cada elemento sin tener que manejar el índice. Es una alternativa al uso de For pudiendo crear códigos más legibles y fácil de mantener.

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

Para la creación de una estructura o struct en C# se debe de seguir la siguiente sintaxis:

```
public struct Coords
{
    public Coords(double x, double y)
    {
        X = x;
        Y = y;
    }

    public double X { get; }
    public double Y { get; }

    public override string ToString() => $"({X}, {Y})";
}
```

Dentro de la estructura Coords se tiene el constructor que recibe los parámetros y los asigna a los atributos de Struct que son “X” e “Y”. Debajo del constructor se declaran las propiedades de la estructura indicando su visibilidad y su getter. El getter permite consultar el valor de las propiedades desde la función Main.

El método de abajo ToString() imprime ambos valores. La palabra reservada override quiere decir que va a reemplazar el método ToString() que proviene de la clase base Object.

24. ¿Qué es una propiedad?

Las propiedades son los campos que modifica al control de usuario, permiten modificar su parte visual o algún comportamiento. Algunos ejemplos de propiedades son; Enabled, Items, Size, BackColor, Font, MinValue, Format, AutoCheck, entre otros.

Las propiedades se pueden modificar en la ventana Propiedades o en el archivo Designer.cs.

25. ¿Qué es un método?

El método es la función que se ejecuta en respuesta a un evento disparado por un control, como un botón. Dentro de este método, se pueden modificar propiedades de otros controles del formulario o del propio control que disparó el evento (para referirse al control actual, se usa la palabra reservada “*this*”).

Los métodos permiten definir el comportamiento de la interfaz en respuesta a la interacción del usuario.

26. ¿Qué es un evento?

Los eventos son acciones que el usuario o el sistema puede provocar, estas acciones ya están predefinidas en el control y algunos ejemplos son; cuando se hace un Click, Enter, DragDrop, DataChanged, MouseDown, MouseUp, entre otros.

Resulta importante realizar la suscripción del evento con un método, esto permite realizar la funcionalidad entre controles y que el usuario pueda ver una interacción con la interfaz de la aplicación.

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

El uso ideal de una variable de tipo boolean es para el retorno de una función. Si por ejemplo se desea validar si se ingresó un String vacío, se puede crear la función validarTexto(String) que retorna un boolean. Al momento de llamar a la función, se puede usar un IF e ir preguntando que valor devolvió la función, de esta forma el código queda más limpio.

Otro uso ideal es hacer un Array de valores booleanos y con el uso de Foreach recorrer este Array y ejecutar un bloque de código cuando es true o false.

Otro uso ideal es complementar un Array de booleanos con un Foreach, de esta forma se puede recorrer al Array y ejecutar un bloque de código cuando se detecta un true o un false en el retorno de la función. El objetivo de usar boolean es mantener un control en el flujo del programa.

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

El método Parse sirve para convertir el tipo de dato String a uno numérico como Double, Decimal o Int. Por ejemplo, si se quiere guardar lo que está en un textBox y después se necesita convertirlo a un Decimal, se hace uso de Parse.

El código para parsear de String a Double es el siguiente:

```
num1 = Decimal.Parse(txtNumero.Text);
```

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

La declaración de un vector de 10 posiciones de tipo Decimal es la siguiente:

```
decimal[] decimales = new decimal[10];
```

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

Cuando se habla de boxing se refiere a convertir un valor de tipo por valor a uno de referencia (por ejemeplo System.Object o System.String). Su conversión es implícita porque el tipo de dato int deriva de la clase Object.

Ejemplo de boxing (conversión implícita):

```
int i = 123;
Object o = i;
```

En cambio, cuando se habla de unboxing se refiere a convertir un valor de referencia a uno de tipo por valor. Su conversión es explicita porque de un Object puede pertenecer a tipos por referencia, por lo tanto, en la conversión se tiene que indicar a qué tipo por valor se le va a asignar a la variable.

Ejemplo de unboxing (conversión explícita):

```
Object o = 123;
int i = (int) o;
```