RESPUESTAS

1. ¿Qué característica posee la programación visual?

La característica principal de la programación visual con C# es el uso de herramientas y componentes visuales para diseñar, desarrollar y depurar aplicaciones. Estas son algunos beneficios de la programación visual:

- ***Interfaz de arrastrar y soltar:*** El entorno de programación visual nos permite diseñar interfaces de usuario y lógica de aplicaciones arrastrando y soltando elementos en un lienzo. Esto incluye controles como botones, cuadros de texto y etiquetas.

- ***Programación basada en eventos:*** C# es un lenguaje basado en eventos y los entornos de programación visual enfatizan este aspecto. Podemos definir fácilmente controladores de eventos para los componentes de la interfaz de usuario, lo que hace que responder a las interacciones del usuario sea intuitivo.

- ***Desarrollo basado en componentes:*** La programación visual con C# implica crear aplicaciones conectando componentes o controles prediseñados. Estos componentes suelen estar encapsulados como objetos con propiedades, métodos y eventos, lo que simplifica el desarrollo de aplicaciones.

- ***Integración con código:*** Nos permite insertar código C# personalizado cuando es necesario. Esto permite la combinación de diseño visual y codificación tradicional para tareas más **complejas.**

2. ¿Qué es un control de usuario?

Un "control de usuario" es un control personalizado reutilizable que podemos crear y utilizar dentro de sus aplicaciones. Nos permite encapsular un grupo de elementos visuales, como botones, etiquetas, cuadros de texto y otros componentes de la interfaz de usuario, junto con la lógica asociada, en un único componente autónomo. Los controles de usuario proporcionan una manera de modularizar y simplificar el diseño y desarrollo de elementos y funcionalidades complejos de la interfaz de usuario dentro de sus aplicaciones C#.

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

Un control de usuario es un componente personalizado diseñado para encapsular un conjunto específico de funcionalidades y elementos de interfaz de usuario. Para crear y utilizar un control de usuario, normalmente consta de tres elementos constituyentes:

- ***Eventos:*** Reacciones que tienen los controles ante distintos contextos. Por ejemplo, una interacción con el usuario por medio de un clic; al momento de cargar la pantalla; al momento de cerrar la pantalla; al cambiar un valor; al cambiar un estado (por ejemplo de “verdadero” a “falso”); Etc.

- ***Comportamiento:*** Los controles de usuario tienen archivos de código subyacente asociados escritos en C#. Este código subyacente contiene la lógica, los controladores de eventos y los métodos que definen el comportamiento y la funcionalidad del control de usuario. Maneja las interacciones del usuario y responde a eventos desencadenados por los elementos visuales.

Propiedades: Son “Variables” que permiten definir el estado de un control, puede ser por ejemplo el color, el tamaño, un valor, el nombre, el texto que muestra en pantalla, etc.

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

El Explorador de soluciones nos muestra las siguientes cosas: Muestra los archivos de/los proyectos de la solución; Permite eliminar y mover los archivos del proyecto; Permite agregar nuevos elementos al proyecto; Establecer referencias a assemblies y servicios web; Crear carpetas; Etc.

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

La ventana de Propiedades permite acceder y modificar a las propiedades y eventos del objeto seleccionado (WebForm, control, clase, etc.)

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

La Caja de herramientas (Toolbox) contiene los componentes y controles que vamos a utilizar en nuestros WebForms

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

Una solución es una estructura de nivel superior que contiene uno o varios proyectos relacionados. Una solución es un contenedor que agrupa proyectos para una aplicación o un conjunto de aplicaciones relacionadas. Puede contener información de configuración global y establecer relaciones entre los diferentes proyectos que lo componen.

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

Un proyecto se encuentra dentro de una solución. Representa una colección de archivos, recursos, código fuente y configuraciones necesarios para construir una parte específica de una aplicación. Podemos entender mejor qué es un proyecto imaginándonos al paquete office, donde cada proyecto es un programa del paquete de Excel, PowerPoint, Word y el paquete office es la solución que contiene todos estos proyectos.

9. ¿Qué es un formulario?

El formulario es con lo que trabajamos en la interfaz visual del proyecto que creamos, en este podemos posicionar los elementos que se encuentran en la caja de herramientas (Toolbox) y a estas dar ciertas propiedades o eventos que hagan durante su ejecucion.

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

Propiedades:

1. ***Text:*** Obtiene o establece el texto que se muestra en el botón.
2. ***Name:*** Obtiene o establece el nombre del control del botón.
3. ***Enabled:*** obtiene o establece un valor que indica si el botón está habilitado para la interacción del usuario.
4. ***Visible:*** Obtiene o establece un valor que indica si el botón es visible en el formulario.
5. ***ForeColor:*** Obtiene o establece el color del texto del botón.
6. ***BackColor:*** Obtiene o establece el color de fondo del botón.
7. ***FlatStyle:*** Obtiene o establece la apariencia de estilo plano del botón (por ejemplo, Plano, Emergente, Estándar).
8. ***Imagen:*** Obtiene o establece una imagen mostrada en el botón.
9. ***ImageAlign:*** Obtiene o establece la alineación de la imagen en relación con el texto del botón.
10. ***AutoSize:*** Obtiene o establece un valor que indica si el botón ajusta automáticamente su tamaño para adaptarse a su contenido.

Métodos:

1. ***PerformClick():*** simula mediante programación un evento de clic en un botón.
2. ***Focus():*** establece el foco de entrada en el control del botón.
3. ***ResetText():*** Restablece el texto del botón a su valor predeterminado.
4. ***ToString():*** Devuelve una representación de cadena del control del botón.
5. ***SendToBack():*** envía el control del botón al final del orden z.
6. ***BringToFront():*** lleva el control del botón al frente del orden z.
7. ***Capture:*** captura la entrada del mouse en el control del botón.
8. ***Hide():*** Oculta el control del botón de la vista.
9. ***Show():*** Muestra el control del botón si estaba previamente oculto.
10. ***Dispose():*** Libera los recursos utilizados por el control del botón.

Eventos:

1. ***Click:*** Ocurre cuando se hace clic en el botón.
2. ***DoubleClick:*** Ocurre cuando se hace doble clic en el botón.
3. ***MouseEnter:*** Ocurre cuando el puntero del mouse ingresa al área de cliente del botón.
4. ***MouseLeave:*** ocurre cuando el puntero del mouse sale del área de cliente del botón.
5. ***MouseHover:*** Ocurre cuando el puntero del mouse pasa sobre el botón.
6. ***MouseDown:*** Ocurre cuando se presiona un botón del mouse mientras el puntero está sobre el botón.
7. ***MouseUp:*** Ocurre cuando se suelta un botón del mouse mientras el puntero está sobre el botón.
8. ***GotFocus:*** ocurre cuando el botón recibe el foco de entrada.
9. ***LostFocus:*** ocurre cuando el botón pierde el foco de entrada.
10. ***EnabledChanged:*** ocurre cuando cambia el valor de la propiedad Enabled del botón.

Estas propiedades, métodos y eventos le permiten personalizar la apariencia, el comportamiento y la funcionalidad de un control Button.

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

Propiedades:

1. ***Checked:*** Obtiene o establece un valor que indica si la casilla de verificación está marcada o desmarcada.
2. ***CheckState:*** obtiene o establece el estado marcado de la casilla de verificación, que puede ser marcado, no marcado o indeterminado.
3. ***Text:*** obtiene o establece el texto que se muestra junto a la casilla de verificación.
4. ***Name:*** Obtiene o establece el nombre del control de casilla de verificación.
5. ***Enabled:*** obtiene o establece un valor que indica si la casilla de verificación está habilitada para la interacción del usuario.
6. ***Visible:*** Obtiene o establece un valor que indica si la casilla de verificación es visible en el formulario.
7. ***ForeColor:*** Obtiene o establece el color del texto de la casilla de verificación.
8. ***BackColor:*** Obtiene o establece el color de fondo de la casilla de verificación.
9. ***ThreeState:*** obtiene o establece un valor que indica si la casilla de verificación admite tres estados (marcado, no marcado, indeterminado).
10. ***AutoCheck:*** Obtiene o establece un valor que indica si al hacer clic en la casilla de verificación se cambiará automáticamente su estado.

Métodos:

1. ***ToString():*** Devuelve una representación de cadena del control de casilla de verificación.
2. ***Focus():*** establece el foco de entrada en el control de casilla de verificación.

3. ***Capture:*** captura la entrada del mouse en el control de casilla de verificación.
4. ***Hide():*** Oculta el control de la casilla de verificación de la vista.
5. ***Show():*** Muestra el control de la casilla de verificación si estaba previamente oculta.
6. ***Dispose():*** Libera los recursos utilizados por el control de casilla de verificación.
7. ***BringToFront():*** lleva el control de la casilla de verificación al frente del orden z.
8. ***SendToBack():*** envía el control de la casilla de verificación al final del orden z.
9. ***OnCheckedChanged(EventArgs e):*** genera el evento CheckedChanged.
10. **OnCheckStateChanged(EventArgs e):** genera el evento CheckStateChanged.

Eventos:

1. ***CheckedChanged:*** ocurre cuando cambia la propiedad Checked de la casilla de verificación.
2. ***CheckStateChanged***: ocurre cuando cambia la propiedad CheckState de la casilla de verificación.
3. ***Click:*** Ocurre cuando se hace clic en la casilla de verificación.
4. ***DoubleClick:*** Ocurre cuando se hace doble clic en la casilla de verificación.
5. ***MouseEnter:*** Ocurre cuando el puntero del mouse ingresa al área de cliente de la casilla de verificación.
6. ***MouseLeave:*** ocurre cuando el puntero del mouse sale del área de cliente de la casilla de verificación.
7. ***MouseHover:*** ocurre cuando el puntero del mouse se coloca sobre la casilla de verificación.
8. ***MouseDown:*** ocurre cuando se presiona un botón del mouse mientras el puntero está sobre la casilla de verificación.
9. ***MouseUp:*** ocurre cuando se suelta un botón del mouse mientras el puntero está sobre la casilla de verificación.
10. ***GotFocus:*** ocurre cuando la casilla de verificación recibe el foco de entrada.

Estas propiedades, métodos y eventos le permiten personalizar la apariencia, el comportamiento y la funcionalidad de un control CheckBox.

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

Propiedades:

1. ***Items:*** obtiene o establece la colección de elementos (cadenas u objetos) en la lista desplegable.
2. ***SelectedIndex:*** obtiene o establece el índice del elemento actualmente seleccionado en la lista.
3. ***SelectedItem:*** Obtiene o establece el elemento seleccionado actualmente en la lista.
4. ***SelectedValue:*** obtiene o establece el valor del elemento seleccionado actualmente (útil cuando se trabaja con controles ComboBox vinculados a datos).
5. ***DropDownStyle:*** Obtiene o establece el estilo del cuadro combinado (DropDown, DropDownList o Simple).
6. ***Text:*** obtiene o establece el texto que se muestra en la parte editable del cuadro combinado.
7. ***Name:*** Obtiene o establece el nombre del control del cuadro combinado.
8. ***Enabled:*** obtiene o establece un valor que indica si el cuadro combinado está habilitado para la interacción del usuario.
9. ***Visible:*** obtiene o establece un valor que indica si el cuadro combinado es visible en el formulario.
10. ***AutoCompleteMode:*** Obtiene o establece el comportamiento de autocompletar del cuadro combinado (por ejemplo, Agregar, Sugerir, Ninguno).

Métodos:

1. ***ToString():*** Devuelve una representación de cadena del control del cuadro combinado.

2. ***Focus():*** establece el foco de entrada en el control del cuadro combinado.
3. ***Hide():*** Oculta el control del cuadro combinado de la vista.
4. ***Show():*** Muestra el control del cuadro combinado si estaba previamente oculto.
5. ***Dispose():*** Libera los recursos utilizados por el control del cuadro combinado.
6. ***BeginUpdate():*** suspende el dibujo del cuadro combinado para mejorar el rendimiento durante las actualizaciones masivas de la colección de elementos.
7. ***EndUpdate():*** Reanuda el dibujo del cuadro combinado después de una serie de actualizaciones iniciadas por BeginUpdate.
8. ***DeselectAll():*** Anula la selección de todos los elementos seleccionados en un cuadro combinado de selección múltiple.
9. ***FindString():*** busca un elemento en la lista que comienza con una cadena especificada y devuelve su índice.
10. ***SelectAll():*** selecciona todos los elementos en un cuadro combinado de selección múltiple.

Eventos:

1. ***SelectedIndexChanged:*** ocurre cuando cambia el índice seleccionado o el elemento seleccionado.
2. ***SelectedValueChanged:*** ocurre cuando cambia el valor seleccionado (útil con controles ComboBox vinculados a datos).
3. ***DropDown:*** ocurre cuando se muestra la parte desplegable del cuadro combinado.
4. **DropDownClosed:** ocurre cuando la parte desplegable del cuadro combinado está cerrada.
5. ***DropDownStyleChanged:*** ocurre cuando cambia la propiedad DropDownStyle.
6. ***TextUpdate:*** ocurre cuando se actualiza la parte de texto del cuadro combinado.
7. ***GotFocus:*** ocurre cuando el cuadro combinado recibe el foco de entrada.
8. ***LostFocus:*** ocurre cuando el cuadro combinado pierde el foco de entrada.
9. ***KeyPress:*** ocurre cuando se presiona una tecla mientras el cuadro combinado tiene el foco.
10. ***MouseClick:*** Ocurre cuando se hace clic en el cuadro combinado con el mouse.

Estas propiedades, métodos y eventos le permiten personalizar el comportamiento y la funcionalidad de un control ComboBox.

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

Propiedades:

1. ***Value:*** Obtiene o establece la fecha y hora actualmente seleccionadas en el control.
2. ***Format:*** obtiene o establece el formato en el que se muestran la fecha y la hora (por ejemplo, corto, largo, personalizado).
3. ***CustomFormat:*** obtiene o establece una cadena de formato de fecha y hora personalizada cuando la propiedad Formato está establecida en Personalizado.
4. ***MinDate:*** Obtiene o establece el valor de fecha mínimo permitido que se puede seleccionar.
5. ***MaxDate:*** Obtiene o establece el valor de fecha máximo permitido que se puede seleccionar.
6. ***ShowUpDown:*** Obtiene o establece un valor que indica si se muestran los botones arriba-abajo para incrementar o disminuir los valores de fecha y hora.
7. ***ShowCheckBox:*** Obtiene o establece un valor que indica si se muestra una casilla de verificación junto al control para habilitar/deshabilitar el control.
8. ***Checked:*** Obtiene o establece el estado marcado de la casilla de verificación (si ShowCheckBox está habilitado).
9. ***CalendarFont:*** Obtiene o establece la fuente utilizada para mostrar el calendario.
10. ***Enebled:*** obtiene o establece un valor que indica si el control está habilitado para la interacción del usuario.

Métodos:

1. ***ToString():*** Devuelve una representación de cadena de la fecha y hora seleccionadas.
2. ***Focus():*** establece el foco de entrada en el control DateTimePicker.
3. ***Hide():*** Oculta el control DateTimePicker de la vista.
4. ***Show():*** Muestra el control DateTimePicker si estaba previamente oculto.
5. ***Dispose():*** Libera los recursos utilizados por el control DateTimePicker.
6. ***ResetText():*** Restablece el texto mostrado en el control al valor predeterminado.
7. ***OnValueChanged(EventArgs e):*** genera el evento ValueChanged.
8. ***OnValueChanged(EventArgs e):*** genera el evento CheckedChanged (si ShowCheckBox está habilitado).
9. ***UpdateDateTimePicker():*** Fuerza una actualización inmediata de la pantalla del control.
10. ***CloseUp():*** cierra el calendario desplegable si está abierto actualmente.

Eventos:

1. ***ValueChanged:*** Ocurre cuando cambia el valor de fecha y hora seleccionada.
2. ***CheckedChanged:*** ocurre cuando cambia la propiedad Checked (si ShowCheckBox está habilitado).
3. ***CloseUp:*** Ocurre cuando el calendario desplegable está cerrado.
4. ***DropDown:*** Ocurre cuando se muestra el calendario desplegable.
5. ***FormatChanged:*** ocurre cuando cambia la propiedad Formato.
6. ***TextChanged:*** Ocurre cuando cambia el texto que se muestra en el control.
7. ***GotFocus:*** ocurre cuando el control DateTimePicker recibe el foco de entrada.
8. ***LostFocus:*** ocurre cuando el control DateTimePicker pierde el foco de entrada.
9. ***KeyPress:*** ocurre cuando se presiona una tecla mientras DateTimePicker tiene el foco.
10. ***MouseClick:*** ocurre cuando se hace clic con el mouse en el control DateTimePicker.

Estas propiedades, métodos y eventos le permiten trabajar con entradas de fecha y hora, personalizar la apariencia y el comportamiento del control DateTimePicker.

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

Propiedades:

1. ***Interval:*** obtiene o establece el tiempo, en milisegundos, entre eventos Timer.Tick.
2. ***Enabled:*** obtiene o establece un valor que indica si el temporizador se está ejecutando.
3. ***Site:*** Obtiene o establece el sitio del control Temporizador.
4. ***Tag:*** Obtiene o establece un objeto que contiene datos sobre el control Temporizador.
5. ***CanRaiseEvents:*** obtiene un valor que indica si el control Timer puede generar eventos.
6. ***DesignMode:*** Obtiene un valor que indica si el control Timer está en modo de diseño.
7. ***Container:*** Obtiene el contenedor para el control Temporizador.
8. ***Modifiers:*** Obtiene o establece el modificador de acceso de la clase del control Timer.
9. ***Start:*** Obtiene o establece la hora a la que debe iniciarse el control Temporizador.
10. ***Stop:*** Obtiene o establece la hora a la que debe detenerse el control del Temporizador.

Métodos:

1. ***Start():*** inicia el control Timer, lo que hace que comience a generar eventos Tick en el intervalo especificado.
2. ***Stop():*** Impide que el control Timer genere eventos Tick.
3. ***ToString():*** Devuelve una representación de cadena del control Temporizador.
4. ***BeginInit():*** Comienza la inicialización del control Timer.
5. ***EndInit():*** Finaliza la inicialización del control Timer.
6. ***Dispose():*** Libera los recursos utilizados por el control Timer.
7. ***OnTick(EventArgs e):*** genera el evento Tick.
8. ***Close():*** Cierra el control Temporizador.

9. ***CreateObjRef():*** crea un objeto que contiene toda la información relevante necesaria para generar un proxy utilizado para comunicarse con un objeto remoto.
10. ***OnEnabledChanged(EventArgs e):*** genera el evento EnabledChanged.

Eventos:

1. ***Tick:*** Ocurre cuando ha transcurrido el intervalo del Temporizador.
2. ***Disposed:*** Ocurre cuando se elimina el control Temporizador.
3. ***EnabledChanged:*** ocurre cuando cambia la propiedad Enabled.
4. ***IntervalChanged:*** ocurre cuando cambia la propiedad Intervalo.
5. ***SiteChanged:*** ocurre cuando cambia la propiedad del sitio.
6. ***TagChanged:*** ocurre cuando cambia la propiedad Tag.
7. ***NameChanged:*** ocurre cuando cambia la propiedad Nombre.
8. ***QueryAccessibilityHelp:*** ocurre cuando una aplicación cliente de accesibilidad consulta la información de accesibilidad para el control Temporizador.
9. ***StyleChanged:*** Ocurre cuando cambia el estilo del control Temporizador.
10. ***Format:*** Este es un evento personalizado que puede implementar si desea formatear la hora o mostrarla de una manera particular cuando el control del Temporizador marca. Puede definir su propia lógica para formatear y suscribirse a este evento para aplicarlo.

Estas propiedades, métodos y eventos proporcionan la funcionalidad necesaria para crear y administrar operaciones basadas en Timer.

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

1 - BackgroundWorker

Propiedades:

1. ***WorkerReportsProgress:*** obtiene o establece un valor que indica si el trabajador en segundo plano puede informar el progreso.
2. ***WorkerSupportsCancellation:*** obtiene o establece un valor que indica si el trabajador en segundo plano admite la cancelación.
3. ***CancellationPending:*** Obtiene un valor que indica si una solicitud de cancelación está pendiente.
4. ***IsBusy:*** obtiene un valor que indica si el trabajador en segundo plano se está ejecutando actualmente.
5. ***Result:*** obtiene el resultado de la operación en segundo plano.

Métodos:

1. ***RunWorkerAsync():*** inicia la operación del trabajador en segundo plano.
2. ***ReportProgress():*** genera el evento ProgressChanged.
3. ***CancelAsync():*** cancela la operación del trabajador en segundo plano.
4. ***RunWorkerCompleted():*** genera el evento RunWorkerCompleted.
5. ***Dispose():*** libera recursos utilizados por el trabajador en segundo plano.

Eventos:

1. ***DoWork:*** Ocurre cuando se ejecuta la operación en segundo plano.
2. ***ProgressChanged:*** ocurre cuando se informa el progreso.
3. ***RunWorkerCompleted:*** ocurre cuando se completa la operación en segundo plano.
4. **Disposed:** ocurre cuando se elimina el trabajador en segundo plano.
5. ***RunWorkerAsyncCompleted:*** ocurre cuando se completa RunWorkerAsync.

2 - CheckedListBox

Propiedades:

1. ***Items:*** Obtiene la colección de artículos en CheckedListBox.
2. ***CheckedItems:*** Obtiene la colección de elementos marcados.
3. ***CheckOnClick:*** obtiene o establece un valor que indica si los elementos se marcan al hacer clic en ellos.
4. ***SelectionMode:*** Obtiene o establece el modo de selección (Uno, Ninguno, MultiSimple, MultiExtended).
5. ***SelectedIndices:*** Obtiene los índices de los elementos seleccionados.

Métodos:

1. ***SetItemChecked():*** establece el estado marcado de un elemento específico.
2. ***GetItemChecked():*** Obtiene el estado marcado de un elemento específico.
3. ***ClearSelected():*** Borra los elementos seleccionados.
4. ***GetSelected():*** Obtiene los elementos seleccionados.
5. ***FindString():*** busca un elemento basándose en una cadena de búsqueda.

Eventos:

1. ***ItemCheck:*** ocurre cuando cambia el estado marcado de un elemento.
2. ***SelectedIndexChanged:*** ocurre cuando cambia el índice seleccionado.
3. ***SelectedValueChanged:*** ocurre cuando cambia el valor seleccionado.
4. ***ItemCheck:*** ocurre cuando cambia el estado marcado de un elemento.
5. ***MouseClick:*** Ocurre cuando se hace clic en el control con el mouse.

3 - ContextMenuStrip

Propiedades:

1. ***Items:*** obtiene la colección de elementos del menú dentro de la franja del menú contextual.
2. ***BackColor:*** Obtiene o establece el color de fondo de la franja del menú contextual.
3. ***ForeColor:*** Obtiene o establece el color de primer plano de la franja del menú contextual.
4. ***Font:*** Obtiene o establece la fuente de la franja del menú contextual.
5. ***Name:*** Obtiene o establece el nombre de la franja del menú contextual.

Métodos:

1. ***Show():*** muestra la franja del menú contextual en una ubicación especificada.
2. ***Hide():*** Oculta la franja del menú contextual.
3. ***Dispose():*** Libera los recursos utilizados por la franja del menú contextual.
4. ***Close():*** Cierra la franja del menú contextual.
5. ***ToString():*** Devuelve una representación de cadena de la franja del menú contextual.

Eventos:

1. ***Opened:*** Ocurre cuando se abre la franja del menú contextual.
2. ***Closed:*** Ocurre cuando la franja del menú contextual está cerrada.
3. ***ItemClicked:*** ocurre cuando se hace clic en un elemento del menú dentro de la franja del menú contextual.
4. ***VisibleChanged:*** Ocurre cuando cambia la visibilidad de la franja del menú contextual.
5. ***Disposed:*** ocurre cuando se elimina la franja del menú contextual.

4 - DataGridView

Propiedades:

1. ***DataSource:*** obtiene o establece la fuente de datos para DataGridView.
2. ***Rows:*** obtiene la colección de filas en DataGridView.
3. ***Columns:*** obtiene la colección de columnas en DataGridView.
4. ***SelectionMode:*** Obtiene o establece el modo de selección (CellSelect, FullRowSelect, etc.).
5. ***AllowUserToAddRows:*** obtiene o establece un valor que indica si los usuarios pueden agregar nuevas filas.

Métodos:

1. ***Refresh():*** actualiza DataGridView para reflejar cualquier cambio en los datos o el diseño.
2. ***ClearSelection():*** Borra la selección actual.
3. ***SelectAll():*** selecciona todas las celdas en DataGridView.
4. ***EndEdit():*** finaliza el modo de edición de la celda actual.
5. ***Sort():*** Ordena el DataGridView por una o más columnas.

Eventos:

1. ***CellClick:*** Ocurre cuando se hace clic en una celda.
2. ***CellValueChanged:*** Ocurre cuando cambia el valor de una celda.
3. ***RowEnter:*** ocurre cuando una nueva fila se convierte en la fila actual.
4. ***DataError:*** ocurre cuando ocurre un error durante el enlace de datos.
5. ***ColumnHeaderMouseClick:*** ocurre cuando se hace clic en el encabezado de una columna.

5 - DataSet

Propiedades:

1. ***Tables:*** Obtiene la colección de tablas dentro del DataSet.
2. ***Relations:*** Obtiene la colección de relaciones entre tablas del DataSet.
3. ***DataSetName:*** Obtiene o establece el nombre del DataSet.
4. ***IsInitialized:*** Obtiene un valor que indica si el DataSet está inicializado.
5. ***HasErrors:*** Obtiene un valor que indica si alguna tabla del DataSet contiene errores.

Métodos:

1. ***Clear():*** borra todos los datos del DataSet.
2. ***AcceptChanges():*** confirma los cambios realizados en el conjunto de datos.
3. ***RejectChanges():*** Rechaza los cambios realizados en el DataSet.
4. ***ReadXml():*** Lee datos XML en el DataSet.
5. ***WriteXml():*** escribe el contenido del conjunto de datos como datos XML.

Eventos:

1. ***Initialized:*** Ocurre cuando se inicializa el DataSet.
2. ***DataError:*** Ocurre cuando ocurre un error durante el procesamiento de datos.
3. ***MergeFailed:*** ocurre cuando falla una operación de combinación.
4. ***Disposed:*** Ocurre cuando se elimina el DataSet.
5. ***Tables.CollectionChanged:*** ocurre cuando cambia la colección de tablas.

6 - GroupBox

Propiedades:

1. ***Text:*** obtiene o establece el texto que se muestra en el cuadro de grupo.
2. ***Name:*** obtiene o establece el nombre del control del cuadro de grupo.
3. ***Enabled:*** obtiene o establece un valor que indica si el cuadro de grupo está habilitado para la interacción del usuario.

4. ***Visible:*** obtiene o establece un valor que indica si el cuadro de grupo es visible en el formulario.
5. ***Font:*** Obtiene o establece la fuente del cuadro de grupo.

Métodos:

1. ***ToString():*** Devuelve una representación de cadena del control del cuadro de grupo.
2. ***Focus():*** establece el foco de entrada en el control del cuadro de grupo.
3. ***Hide():*** Oculta el control del cuadro de grupo de la vista.
4. ***Show():*** Muestra el control del cuadro de grupo si estaba previamente oculto.
5. ***Dispose():*** Libera los recursos utilizados por el control del cuadro de grupo.

Eventos:

1. ***DoubleClick:*** ocurre cuando se hace doble clic en el cuadro de grupo.
2. ***Click:*** ocurre cuando se hace clic en el cuadro de grupo.
3. ***GotFocus:*** ocurre cuando el cuadro de grupo recibe el foco de entrada.
4. ***LostFocus:*** ocurre cuando el cuadro de grupo pierde el foco de entrada.
5. ***TextChanged:*** ocurre cuando cambia la propiedad Texto.

7 - HScrollBar

Propiedades:

1. ***Value:*** obtiene o establece el valor actual de la barra de desplazamiento horizontal.
2. ***Minimun:*** Obtiene o establece el valor mínimo de la barra de desplazamiento.
3. ***Maximum:*** Obtiene o establece el valor máximo de la barra de desplazamiento.
4. ***SmallChange:*** obtiene o establece la cantidad para incrementar o disminuir el valor cuando el usuario hace clic en las flechas.
5. ***LargeChange:*** obtiene o establece la cantidad para incrementar o disminuir el valor cuando el usuario hace clic en la pista de la barra de desplazamiento.

Métodos:

1. ***ToString():*** Devuelve una representación de cadena del control de la barra de desplazamiento horizontal.
2. ***Focus():*** establece el foco de entrada en el control de la barra de desplazamiento horizontal.
3. ***Hide():*** Oculta el control de la barra de desplazamiento horizontal de la vista.
4. ***Show():*** Muestra el control de la barra de desplazamiento horizontal si estaba previamente oculto.
5. ***Dispose():*** Libera los recursos utilizados por el control de la barra de desplazamiento horizontal.

Eventos:

1. ***Scroll:*** Ocurre cuando cambia el valor de la barra de desplazamiento.
2. ***ValueChanged:*** ocurre cuando cambia la propiedad Valor.
3. ***MouseClick:*** Ocurre cuando se hace clic con el mouse en la barra de desplazamiento.
4. ***GotFocus:*** ocurre cuando la barra de desplazamiento recibe el foco de entrada.
5. ***LostFocus:*** ocurre cuando la barra de desplazamiento pierde el foco de entrada.

8 - ImageList

Propiedades:

1. ***Images:*** Obtiene la colección de imágenes en la lista de imágenes.

2. ***ColorDepth:*** Obtiene o establece la profundidad de color de las imágenes (Profundidad4Bit, Profundidad8Bit, Profundidad16Bit, Profundidad24Bit, Profundidad32Bit).
3. ***ImageSize:*** Obtiene o establece el tamaño de las imágenes en la lista de imágenes.
4. ***TransparentColor:*** Obtiene o establece el color transparente de las imágenes.
5. ***ImageStream:*** obtiene o establece la secuencia de imágenes para la lista de imágenes.

Métodos:

1. ***ToString():*** Devuelve una representación de cadena del control de lista de imágenes.
2. ***Add():*** Agrega una imagen a la lista de imágenes.
3. ***Clear():*** borra todas las imágenes de la lista de imágenes.
4. ***Remove():*** Elimina una imagen de la lista de imágenes.
5. ***Dispose():*** Libera los recursos utilizados por el control de lista de imágenes.

Eventos:

1. ***RecreateHandle:*** ocurre cuando se recrea el identificador de la lista de imágenes.
2. ***Disposed:*** Ocurre cuando se elimina la lista de imágenes.
3. ***ChangeUICues:*** ocurre cuando cambian las señales de la interfaz de usuario.
4. ***ImagesChanged:*** ocurre cuando cambian las imágenes en la lista de imágenes.
5. ***PaletteChanged:*** Ocurre cuando cambia la paleta del sistema.

9 - LinkLabel

Propiedades:

1. ***Text:*** obtiene o establece el texto que se muestra en la etiqueta del enlace.
2. ***Links:*** obtiene la colección de enlaces dentro de la etiqueta del enlace.
3. ***LinkBehavior:*** Obtiene o establece el comportamiento de los enlaces (SystemDefault, AlwaysUnderline, HoverUnderline, NeverUnderline).
4. ***ActiveLinkColor:*** Obtiene o establece el color de los enlaces activos.
5. ***VisitedLinkColor:*** Obtiene o establece el color de los enlaces visitados.

Métodos:

1. ***ToString():*** Devuelve una representación de cadena del control de etiqueta de enlace.
2. ***Focus():*** establece el foco de entrada en el control de etiqueta del enlace.
3. ***Hide():*** Oculta el control de etiqueta de enlace de la vista.
4. ***Show():*** Muestra el control de la etiqueta del enlace si estaba previamente oculto.
5. ***Dispose():*** Libera los recursos utilizados por el control de etiqueta de enlace.

Eventos:

1. ***LinkClicked:*** Ocurre cuando se hace clic en un enlace dentro de la etiqueta del enlace.
2. ***GotFocus:*** ocurre cuando la etiqueta del enlace recibe el foco de entrada.
3. ***LostFocus:*** ocurre cuando la etiqueta del enlace pierde el foco de entrada.
4. ***MouseEnter:*** Ocurre cuando el puntero del mouse ingresa a la etiqueta del enlace.
5. ***MouseLeave:*** ocurre cuando el puntero del mouse sale de la etiqueta del enlace.

10 - MenuStrip

Propiedades:

1. ***Items:*** obtiene la colección de elementos del menú dentro de la franja del menú.
2. ***BackColor:*** Obtiene o establece el color de fondo de la franja del menú.
3. ***ForeColor:*** Obtiene o establece el color de primer plano de la franja del menú.
4. ***Font:*** Obtiene o establece la fuente de la franja del menú.

5. ***Name:*** Obtiene o establece el nombre de la franja del menú.

Métodos:

1. ***ToString():*** Devuelve una representación de cadena del control de franja de menú.
2. ***Focus():*** establece el foco de entrada en el control de la barra de menú.
3. ***Hide():*** Oculta el control de la barra de menú de la vista.
4. ***Show():*** Muestra el control de la barra de menú si estaba previamente oculto.
5. ***Dispose():*** Libera los recursos utilizados por el control de la barra de menú.

Eventos:

1. ***ItemClicked:*** ocurre cuando se hace clic en un elemento del menú dentro de la franja del menú.
2. ***MenuActivate:*** Ocurre cuando se activa la franja del menú.
3. ***MenuDeactivate:*** Ocurre cuando la franja del menú está desactivada.
4. ***Paint:*** Ocurre cuando se pinta la franja del menú.
5. ***Disposed:*** Ocurre cuando se elimina la franja del menú.

16. ¿Cómo se declara un procedimiento? Ejemplo.

Un procedimiento se declara de la siguiente manera: ***void nombre (variable*)** Tiene la particularidad de no poseer return por lo tanto no retorna valores

17. ¿Cómo se declara una función? Ejemplo

Una función se declara ***TipoDato nombre (variable)*** Tiene la particularidad de utilizar return y devolver valores del tipo de dato definido previamente

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

La diferencia que existe entre una función y un procedimiento radica principalmente en el uso del **return.** Otra diferencia es que las funciones se declaran según el tipo de dato que devuelven y los procedimientos se declaran con **void**

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

Cuando pasamos un parámetro por valor estamos estableciendo el valor de entrada del parametro de dicho procedimiento como una copia del valor original, y cuando termine de ejecutarse el procedimiento el valor del parametro si cambio dentro del procedimiento en el parametro original seguira siendo el mismo.

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

Cuando pasamos un parámetro por valor estamos estableciendo el valor de la entrada del parametro de dicho procedimiento sera la direccion de memoria del parametro original, y cuando termine de ejecutarse el procedimiento el valor del parametro si cambio dentro del procedimiento entonce el parametro original tambien sera modificado.

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

Las estructuras de desicion son 3:

IF

int edad = 18;

```
if (edad >= 18) {
        Console.WriteLine("Eres mayor de edad.");
}
```

IF - ELSE

```
int nota = 85;
if (nota >= 60)
        Console.WriteLine("Aprobado");
else:
        Console.WriteLine("Reprobado");
```

SWITCH

```
char operador = '+';
switch (operador) {
        case '+':
                Console.WriteLine("Suma");
                break;
        case '-':
                Console.WriteLine("Resta");
                break;
        default:
                Console.WriteLine("Operador desconocido");
                break;
}
```

Las principales diferencias entre estas estructuras radican en cómo manejan las condiciones y en la cantidad de opciones. If y if-else son adecuados para manejar una o dos condiciones y Switch es más apropiado cuando se deben considerar múltiples casos. Cada estructura tiene su lugar en diferentes situaciones, y la elección dependerá del problema que resolvamos.

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

FOR: Este bucle se utiliza cuando se conoce la cantidad exacta de iteraciones.

```
for (int i = 0; i < 5; i++) {
        Console.WriteLine($"Iteración #{i}");
}
```

WHILE: Este bucle se utiliza cuando no sabemos la cantidad exacta de iteraciones que necesitamos

```
int count = 0;

while (count < 5) {

        Console.WriteLine($"Iteración #{count}");

        count++;

}
```

DO WHILE: Similar al bucle while, pero se verifica la condición después de cada iteración. Esto garantiza que el bucle se ejecute al menos una vez.

```
int num = 5;

do {

        Console.WriteLine($"Número: {num}");

        num--;

} while (num > 0);
```

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

La estructura se crea de la siguiente forma:

```
namespace EjemploEstructura {

        // Declaración de la estructura

        struct Persona     {

                public string Nombre;

                public int Edad;

        }

        class Program     {

                static void Main(string[] args)         {

                        // Crear una instancia de la estructura

                        Persona persona1;

                        persona1.Nombre = "Juan";

                        persona1.Edad = 30;

                        // Acceder a los campos de la estructura

                        Console.WriteLine($"Nombre: {persona1.Nombre}");

                        Console.WriteLine($"Edad: {persona1.Edad}");
```

```
            }
        }
}
```

24. ¿Qué es una propiedad?

La propiedad proporciona un mecanismo para poder acceder a la modificación de datos encapsulados

25. ¿Qué es un método?

Método / Función / Procedimiento, a menudo son utilizados como sinónimos pero no es correcto del todo, a grandes rasgos podemos decir que un método es una porción de código reutilizable que realiza una tarea especifica.

26. ¿Qué es un evento?

Los eventos son comunes en la programación visual, ya que permiten que los controles interactúen con el usuario y respondan a las acciones del usuario

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

El uso ideal de un tipo Booleana es cuando lo utilizamos para controlar el flujo de una estructura de repetición, crear interrupciones o validar condiciones

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

El método Parse en este caso **“Decimal.Parse”** se utiliza para las conversiones de un tipo de dato al dato seleccionado previamente del “.Parse “, es importante saber que también existe la función **“ Convert.Tipo ”,** pero tienen utilizaciones distintas y cada una interactúa de manera distinta con el dato **NULL.**

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

La forma de declarar un vertor de 10 posiciones del tipo decimal es: **decimal[ ] num = new decimal[10];**

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

Boxing, es convertir un tipo de valor a referencia para ser almacenado en un object.

UnBoxing, es convertir un tipo de referencia en un tipo de valor. Lo más relevante de estos términos es el rendimiento y como afecta en términos de sobrecarga a una aplicación o a nuestro trabajo.