Herramientas de Programación 1

Tema Nº7:

Estructuras

Indicador de logro Nº7:Aplica las estructuras de decisión e iterativas, subrutinas y estructuras de datos (array, colecciones) en el lenguaje C# para el desarrollo aplicaciones de alto rendimiento

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº7:**

ESTRUCTURAS DE DECISIÓN E ITERATIVAS

Estructura De Decisión: If

Las estructuras de decisión o selectivas son aquellas que nos permiten hacer una selección entre dos o varias rutas de ejecución posibles. La selección se llevará a cabo según el valor de una expresión. Esta expresión puede ser una expresión relacional.

Nuestra primera estructura selectiva se conoce como if, que es un si condicional. Si tal cosa sucede, entonces haz tal cosa. El uso del if es sencillo:

**Ejemplos:**

if(tieneDescuento && precio >= 100)

{

montoDescuento = precio \* porcentajeDescuento / 100;

total = precio - montoDescuento;

}

Es bastante útil ejecutar un conjunto de instrucciones si es que la condición no se cumple. Para lograr esto haremos uso de else. Siempre utilizaremos else a continuación de una sentencia if, ya que else no se puede usar sólo. Tenemos que colocar nuestro código de la siguiente forma:

**Ejemplos:**

if (codigoDescuento == "201" || codigoDescuento == "202" || codigoDescuento == "203")

{

porcentajeDescuento = 20;

}

else

{

porcentajeDescuento = 10;

}

En las estructuras de decisión y las iterativas, si la cantidad de instrucciones dentro de las llaves será de una sola, no es necesario poner las llaves, entonces, el código anterior funcionara correctamente si lo escribimos de la siguiente forma:

**Ejemplos:**

if (codigoDescuento == "201" || codigoDescuento == "202" || codigoDescuento == "203")

porcentajeDescuento = 20;

else

porcentajeDescuento = 10;

Existe una variante de la estructura if que nos permite evaluar varias condiciones a la vez sin necesidad de ejecutar varios if independientemente, lo explicamos con el siguiente

**Ejemplos:**

if (codigoDescuento == "201")

porcentajeDescuento = 20;

else if (codigoDescuento == "202")

porcentajeDescuento = 15;

else if (codigoDescuento == "203")

porcentajeDescuento = 10;

else

porcentajeDescuento = 0;

Si la condición del if no se cumple y tampoco de los else if se, ejecutarán las sentencias del else.

Estructura de Decisión: Switch

En ejemplos anteriores hemos observado que para cada operación debemos utilizar un if. Esto es correcto, y en varias ocasiones veremos que es necesario tomar diferentes opciones dependiendo del valor de una variable.

El valor de la variable se compara con un valor para cada caso. Si el valor coincide, entonces se empieza a ejecutar el código a partir de esa línea.

**Ejemplos:**

switch (num)

{

case 1:

numero = "uno";

break;

case 2:

numero = "dos";

break;

case 3:

numero = "tres";

break;

case 4:

numero = "cuatro";

break;

case 5:

numero = "cinco";

break;

default:

numero = "error";

break;

}

Cuando usamos switch es necesario colocar entre paréntesis la variable que utilizaremos para llevar a cabo las comparaciones. Luego tenemos que crear un bloque de código y colocar adentro de él los casos y el código a ejecutar para cada caso. Para indicar un caso, usamos case seguido del valor de comparación y dos puntos.

Existe un caso llamado default, que podemos utilizar si lo deseamos. Este caso siempre debe ser el último caso definido.

Si no hacemos uso de break y el caso está vacío, entonces el programa continuará ejecutando el código del próximo caso y así sucesivamente hasta el final del bloque del código que pertenece al switch.

Estructura Iterativa: For

Antes de resolver cualquier problema con el ciclo for, lo primero que tenemos que hacer es aprender a utilizar el ciclo. La sentencia del ciclo se inicia con la palabra clave for seguida de paréntesis. Adentro de los paréntesis colocaremos expresiones que sirven para controlar el ciclo. El ciclo for tiene cuatro partes principales:

for(inicializacion; condicion; incremento)

código

En primer lugar encontramos la inicialización. En esta sección le damos a la variable de control su valor inicial. El valor inicial se lleva a cabo por medio de una asignación normal, tal y como las que hemos trabajado. La segunda sección lleva una condición en forma de una expresión relacional.

Estructura Iterativa: While

El ciclo while también puede ser utilizado cuando tenemos algo que se debe repetir pero no conocemos el número de repeticiones previamente. La repetición del ciclo tiene que ver con el cumplimiento de una condición. Este ciclo puede no ejecutarse ni siquiera una vez. Su estructura es la siguiente:

while(condición)

código

Estructura Do While

El ciclo do while permite que cierto código se repita mientras una condición se evalúe como verdadera. El valor de la evaluación dependerá del estatus del programa en un momento dado.

El ciclo do while se codifica de la siguiente manera:

do

{

código

}(condición);

**Ejemplos:**

**RECURSOS**

1. Hardware

* Una computadora con 4 GB e RAM y procesador de 2.0 GHz.

1. Software

* Visual Studio Professional 2015 / 2017

**PROCEDIMIENTO**

**Laboratorio 7:**

7.1 Estructuras de decisión if y switch.

**Objetivos**

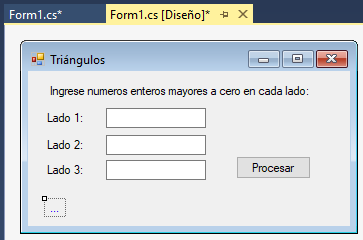
* Desarrollar aplicaciones haciendo uso de las estructuras de decisión if y switch.
* Implementar métodos dentro de una Clase.
* Implementar Clases y Métodos Estáticos.

**Introducción**

Durante esta actividad, vamos aprender a tomar decisiones en el flujo de la ejecución de nuestras aplicaciones haciendo uso de las estructuras de decisión if y switch.

Paso 1. Estructura de decisión if.

1. Creamos el proyecto Semana07, el formulario lo diseñamos de acuerdo a:



|  |  |  |
| --- | --- | --- |
| Control | Propiedad | Valor |
| Form1 | (Name) | frmTriangulos |
|  | Text | Triángulos |
| TextBox1 | (Name) | txtLado1 |
| TextBox2 | (Name) | txtLado2 |
| TextBox3 | (Name) | txtLado3 |
| Button1 | (Name) | btnProcesar |
|  | Text | Procesar |
| Label1 | (Name) | lblResultdo |
|  | ForeColor | Blue [Web] |

1. Agregamos código al evento Click del botón:

private void btnProcesar\_Click(object sender, EventArgs e)

{

int lado1 = Convert.ToInt32(txtLado1.Text);

int lado2 = Convert.ToInt32(txtLado2.Text);

int lado3 = Convert.ToInt32(txtLado3.Text);

string resultado = "";

if((lado1 >= lado2 + lado3) ||

(lado2 >= lado1 + lado3) ||

(lado3 >= lado1 + lado2))

{

resultado = "No es un triángulo";

}

else

{

if (lado1 == lado2 && lado2 == lado3)

resultado = "EQUILATERO";

else if (lado1 == lado2 || lado2 == lado3 || lado1 == lado3)

resultado = "ISOSCELES";

else

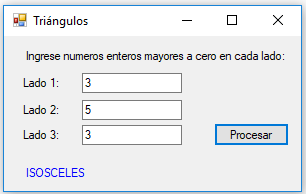
resultado = "ESCALENO";

}

lblResultdo.Text = resultado;

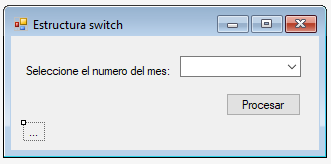
}

1. Ejecutamos la aplicación.



Paso 2. Estructura de decisión switch.

1. En el proyecto Semana07 creamos un nuevo formulario con el siguiente diseño:



|  |  |  |
| --- | --- | --- |
| Control | Propiedad | Valor |
| Form2 | (Name) | frmSwitch |
|  | Text | Estructura switch |
| ComboBox1 | (Name) | cmbMes |
| Label2 | (Name) | lblResultado |
| Button1 | (Name) | btnProcesar |
|  | Text | Procesar |

1. Agregamos código en el evento Load del formulario para agregar los ítems del ComboBox:

private void frmSwitch\_Load(object sender, EventArgs e)

{

for(int i = 1; i < 13; i++)

cmbMes.Items.Add(i);

}

1. Agregamos código al evento Click del botón:

private void btnProcesar\_Click(object sender, EventArgs e)

{

int numeroMes = 0;

if (cmbMes.Text.Length > 0)

{

numeroMes = Convert.ToInt32(cmbMes.Text);

lblResultado.ForeColor = Color.Blue;

switch (numeroMes)

{

case 1:

lblResultado.Text = "Enero";

break;

case 2:

lblResultado.Text = "Febrero";

break;

case 3:

lblResultado.Text = "Marzo";

break;

case 4:

lblResultado.Text = "Abril";

break;

case 5:

lblResultado.Text = "Mayo";

break;

case 6:

lblResultado.Text = "Junio";

break;

case 7:

lblResultado.Text = "Julio";

lblResultado.ForeColor = Color.Green;

break;

case 8:

lblResultado.Text = "Agosto";

lblResultado.ForeColor = Color.Green;

break;

case 9:

lblResultado.Text = "Setiembre";

lblResultado.ForeColor = Color.Green;

break;

case 10:

lblResultado.Text = "Octubre";

lblResultado.ForeColor = Color.Green;

break;

case 11:

lblResultado.Text = "Noviembre";

lblResultado.ForeColor = Color.Green;

break;

case 12:

lblResultado.Text = "Diciembre";

lblResultado.ForeColor = Color.Green;

break;

}

}

else

{

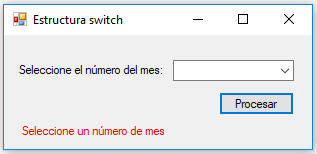
lblResultado.Text = "Seleccione un número de mes";

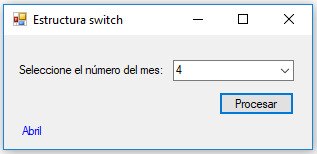
lblResultado.ForeColor = Color.Red;

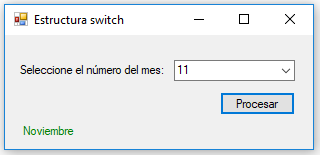
}

}

1. Ejecutamos la aplicación.







**6.2 Estructuras iterativas for, While y do while.**

**Objetivos**

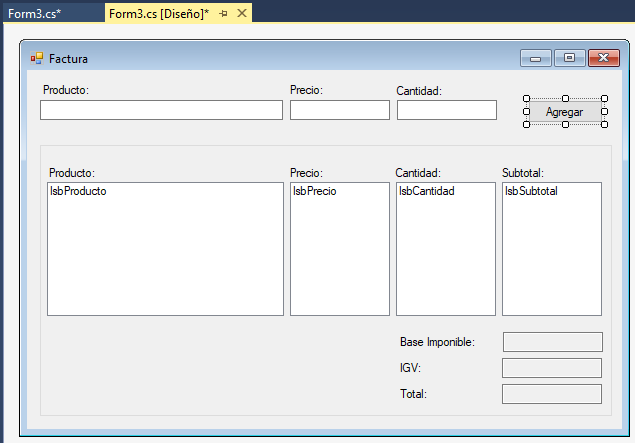
* Desarrollar aplicaciones haciendo uso de las estructuras iterativas for, While y do while.

**Introducción**

Durante esta actividad, vamos aprender a ejecutar cíclicamente un grupo de instrucciones de acuerdo a los requerimientos que se necesitan cumplir en los programas.

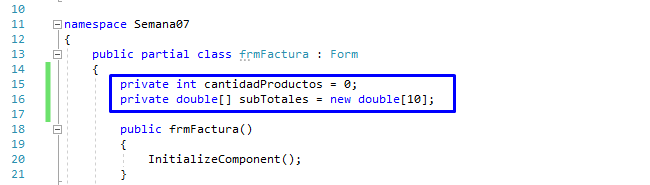
Paso 1. Estructura iterativa for.

1. creamos un nuevo formulario con el siguiente diseño:



|  |  |  |
| --- | --- | --- |
| Control | Propiedad | Valor |
| Form3 | (Name) | frmFactura |
|  | Text | Factura |
| TextBox1 | (Name) | txtProducto |
| TextBox1 | (Name) | txtPrecio |
| TextBox1 | (Name) | txtCantidad |
| TextBox1 | (Name) | txtBaseImponible |
|  | ReadOnly | True |
| TextBox1 | (Name) | txtIgv |
|  | ReadOnly | True |
| TextBox1 | (Name) | txtTotal |
|  | ReadOnly | True |
| ListBox1 | (Name) | lsbProducto |
| ListBox1 | (Name) | lsbPrecio |
| ListBox1 | (Name) | lsbCantidad |
| ListBox1 | (Name) | lsbSubtotal |
| Button1 | (Name) | btnAgregar |

Vamos a crear dos variables (atributos) dentro de la clase del formulario, estos nos servirán para contabilizar la cantidad de productos agregados y sumar los subtotales.



A estos dos atributos se puede acceder desde cualquier manejador de eventos del mismo formulario.

1. Agregamos código al evento Click del botón de acuerdo a:

private void btnAgregar\_Click(object sender, EventArgs e)

{

double precio = Convert.ToDouble(txtPrecio.Text);

double cantidad = Convert.ToDouble(txtCantidad.Text);

double subtotal = precio \* cantidad;

double baseImponible = 0;

double montoIgv = 0;

double montoTotal = 0;

cantidadProductos++;

if (cantidadProductos < 11)

{

lsbProducto.Items.Add(txtProducto.Text);

lsbPrecio.Items.Add(txtPrecio.Text);

lsbCantidad.Items.Add(txtCantidad.Text);

lsbSubtotal.Items.Add(subtotal.ToString());

subTotales[cantidadProductos - 1] = subtotal;

for (int i = 0; i < cantidadProductos; i++)

{

baseImponible = baseImponible + subTotales[i];

}

montoIgv = baseImponible \* 18 / 100;

montoTotal = baseImponible + montoIgv;

txtBaseImponible.Text = baseImponible.ToString();

txtIgv.Text = montoIgv.ToString();

txtTotal.Text = montoTotal.ToString();

txtProducto.Text = "";

txtPrecio.Text = "";

txtCantidad.Text = "";

txtProducto.Focus();

}

else

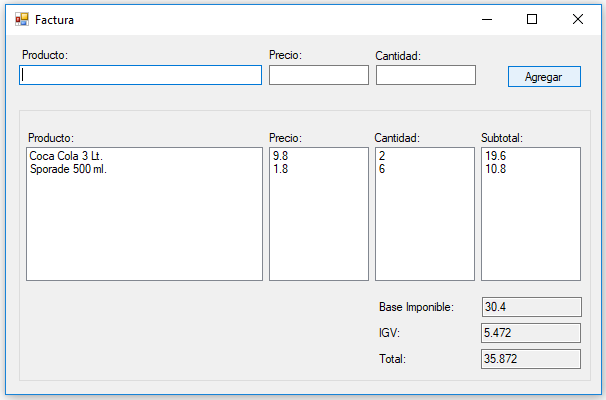
{

MessageBox.Show("Son 10 productos como máximo.", "Lleno");

}

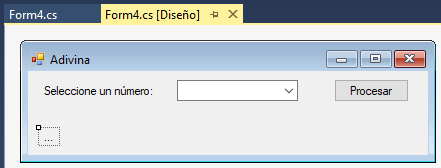
}

1. Ejecutamos la aplicación.



Paso 2. Estructura iterativa while.

1. Creamos un nuevo formulario con el siguiente diseño.



|  |  |  |
| --- | --- | --- |
| Control | Propiedad | Valor |
| Form4 | (Name) | frmSuma |
|  | Text | Suma de los primeros 'n' numeros enteros |
| ComboBox1 | (Name) | cmbNumeros |
| Button1 | (Name) | btnProcesar |
|  | Text | Procesar |
| Label2 | (Name) | lblMensaje |
|  | Text | … |
|  | ForeColor | Blue [Web] |

1. Para cargar los ítems el ComboBox vamos agregar código al evento Load del formulario.

private void frmSuma\_Load(object sender, EventArgs e)

{

int contador = 1;

while (contador < 21)

{

cmbNumeros.Items.Add(contador.ToString());

contador++;

}

cmbNumeros.SelectedIndex = 0;

}

1. Para calcular la suma de los ‘n’ primeros números vamos agregar el siguiente código en el evento Click del botón.

private void btnProcesar\_Click(object sender, EventArgs e)

{

int numeroSelecionado = Convert.ToInt32(cmbNumeros.SelectedItem.ToString());

string mensaje = "La suma de los " +

numeroSelecionado.ToString() +

" primeros números es: ";

int suma = 0;

int contador = 1;

while(contador <= numeroSelecionado)

{

suma = suma + contador;

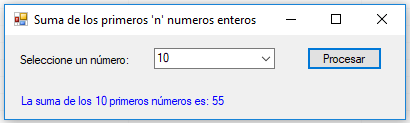
contador++;

}

lblMensaje.Text = mensaje + suma.ToString();

}

1. Ejecutamos la aplicación.



Paso 3. Estructura iterativa do while.

Vamos a realizar el mismo ejemplo anterior pero ahora usando do While.

1. Modificamos el código de los dos eventos.

private void frmSuma\_Load(object sender, EventArgs e)

{

int contador = 0;

do

{

contador++;

cmbNumeros.Items.Add(contador.ToString());

} while (contador < 20);

cmbNumeros.SelectedIndex = 0;

}

private void btnProcesar\_Click(object sender, EventArgs e)

{

int numeroSelecionado = Convert.ToInt32(cmbNumeros.SelectedItem.ToString());

string mensaje = "La suma de los " +

numeroSelecionado.ToString() +

" primeros números es: ";

int suma = 0;

int contador = 0;

do

{

contador++;

suma = suma + contador;

} while (contador < numeroSelecionado) ;

lblMensaje.Text = mensaje + suma.ToString();

}

1. Ejecutamos la aplicación y validamos que cumple con la misma funcionalidad.

**METODOS EN C#**

Un método es un bloque de código que contiene una serie de instrucciones. Un programa hace que se ejecuten las instrucciones al llamar al método y especificando los argumentos de método necesarios.

Los métodos se declaran en una [clase](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/class) o [struct](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/struct) especificando el nivel de acceso, como public o private, modificadores opcionales como static, abstract o sealed, el valor de retorno, el nombre del método y cualquier parámetro de método. Todas estas partes forman la firma del método.

private double Sumar(double pNumero1, double pNumero2)

{

double resultado = pNumero1 + pNumero2;

return resultado;

}

Los parámetros de método se encierran entre paréntesis y se separan por comas. Los paréntesis vacíos indican que el método no requiere parámetros

Llamar a un método en un objeto es como acceder a un campo. Después del nombre del objeto, agregue un punto, el nombre del método y paréntesis. Los argumentos se enumeran entre paréntesis y están separados por comas.

double resultado = Sumar(numero1, numero2);

Los métodos pueden devolver un valor al autor de llamada. Si el tipo de valor devuelto, el tipo enumerado antes del nombre de método, no es void, el método puede devolver el valor mediante la utilización de la palabra clave return Una instrucción con la palabra clave return seguida de un valor que coincide con el tipo de valor devuelto devolverá este valor al autor de llamada del método.

**CLASES Y METODOS ESTATICOS**

En C# podemos definir métodos que se crean independientemente a la definición de objetos. Un método estático puede llamarse sin tener que crear un objeto o instancia de dicha clase. Un método estático tiene ciertas restricciones:

* No puede acceder a los atributos de la clase (salvo que sean estáticos)
* No puede utilizar el operador this, ya que este método se puede llamar sin tener que crear un objeto de la clase.
* Puede llamar a otro método siempre y cuando sea estático.
* Un método estático es lo más parecido a lo que son las funciones en los lenguajes estructurados (con la diferencia que se encuentra encapsulado en una clase)

**LAS COLECCIONES**

Las colecciones son estructuras de datos que nos permiten guardar en su interior cualquier tipo de información. Existen diferentes tipos de colecciones y la forma como se guarda, se accede y se elimina la información en cada una de ellas es distinta. En los arreglos nosotros teníamos que indicar la cantidad de elementos que el arreglo debía tener. En las colecciones esto no es necesario, ya que es posible agregar elementos dinámicamente. Esto quiere decir que cuando el programa se está ejecutando podemos adicionar o borrar sus elementos.

Las colecciones que brinda C# son los siguientes:

* ArrayList
* Hashtable
* Queue
* Stack.

La colección que aprenderemos en este capítulo es: ArrayList.

**ARRAYLIST**

Guarda la información como si fuera una lista. Y sobre esta lista es posible realizar diferentes actividades con los elementos almacenados. Entendemos al ArrayList como un arreglo que puede cambiar su tamaño según lo necesitemos. El ArrayList tiene una propiedad que llamamos capacidad, que indica el tamaño que ocupa la lista. También tenemos el conteo, el cual nos dice cuántos elementos está guardando en su interior.

La declaración se lleva a cabo de la siguiente manera:

ArrayList datos = new ArrayList();

Lo primero que necesitamos es indicar ArrayList, ya que éste es el nombre de la clase. Luego colocamos el nombre que va a tener, en nuestro caso es datos. En este ejemplo el constructor de la clase no recibe ningún parámetro. Si bien el ArrayList aumenta su tamaño dinámicamente, es posible instanciar el arreglo con algún valor de capacidad inicial. Esto es útil si sabemos inicialmente cuantos elementos puede contener el ArrayList. para hacerlo simplemente colocamos la capacidad inicial entre los paréntesis de la siguiente forma:

ArrayList datos = new ArrayList(32);

Aquí, datos tiene una capacidad inicial de 32, aunque se encuentre vacío.

Esto resulta muy útil cuando sabemos que debemos recorrer todos los elementos de la estructura, como un ArrayList, y hacer algo con ellos.

foreach(int valor in costo)

{

lsbCantidad.Items.Add(valor.ToString());

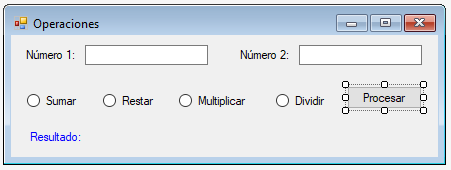
}

**Métodos de Clase. Clases y Métodos Estáticos.**

Durante esta actividad, vamos aprender a modular nuestras aplicaciones haciendo uso de Clases y Métodos Estáticos.

Paso 1. Métodos de Clase.

1. Creamos el proyecto Semana08, el formulario lo diseñamos de acuerdo a:



|  |  |  |
| --- | --- | --- |
| Control | Propiedad | Valor |
| Form1 | (Name) | frmOperaciones |
|  | Text | Operaciones |
| TextBox1 | (Name) | txtNumero1 |
| TextBox2 | (Name) | txtNumero2 |
| RadioButton1 | (Name) | rbtSumar |
|  | Text | Sumar |
| RadioButton2 | (Name) | rbtRestar |
|  | Text | Restar |
| RadioButton3 | (Name) | rbtMultiplicar |
|  | Text | Multiplicar |
| RadioButton4 | (Name) | rbtDividir |
|  | Text | Dividir |
| Button1 | (Name) | btnProcesar |
|  | Text | Procesar |
| Label1 | (Name) | lblResultdo |
|  | Text | Resultdo |
|  | ForeColor | Blue [Web] |

1. Agregamos código al evento Load del formulario:

private void frmOperaciones\_Load(object sender, EventArgs e)

{

rbtSumar.Checked = true;

}

1. Agregamos cuatro métodos a la clase del formulario frmOperaciones:

private double Sumar(double pNumero1, double pNumero2)

{

double resultado = pNumero1 + pNumero2;

return resultado;

}

private double Restar(double pNumero1, double pNumero2)

{

double resultado = pNumero1 - pNumero2;

return resultado;

}

private double Multiplicar(double pNumero1, double pNumero2)

{

double resultado = pNumero1 \* pNumero2;

return resultado;

}

private double Dividir(double pNumero1, double pNumero2)

{

double resultado = pNumero1 / pNumero2;

return resultado;

}

Agregamos código al evento Click del botón:

private void btnProcesar\_Click(object sender, EventArgs e)

{

double numero1 = Convert.ToDouble(txtNumero1.Text);

double numero2 = Convert.ToDouble(txtNumero2.Text);

double resultado = 0;

string operacion = "";

if (rbtSumar.Checked)

operacion = rbtSumar.Text;

else if (rbtRestar.Checked)

operacion = rbtRestar.Text;

else if (rbtMultiplicar.Checked)

operacion = rbtMultiplicar.Text;

else if (rbtDividir.Checked)

operacion = rbtDividir.Text;

switch (operacion)

{

case "Sumar":

resultado = Sumar(numero1, numero2);

break;

case "Restar":

resultado = Restar(numero1, numero2);

break;

case "Multiplicar":

resultado = Multiplicar(numero1, numero2);

break;

case "Dividir":

resultado = Dividir(numero1, numero2);

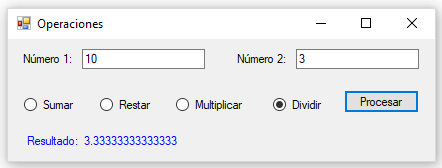
break;

}

lblResultado.Text = "Resultado: " + resultado.ToString();

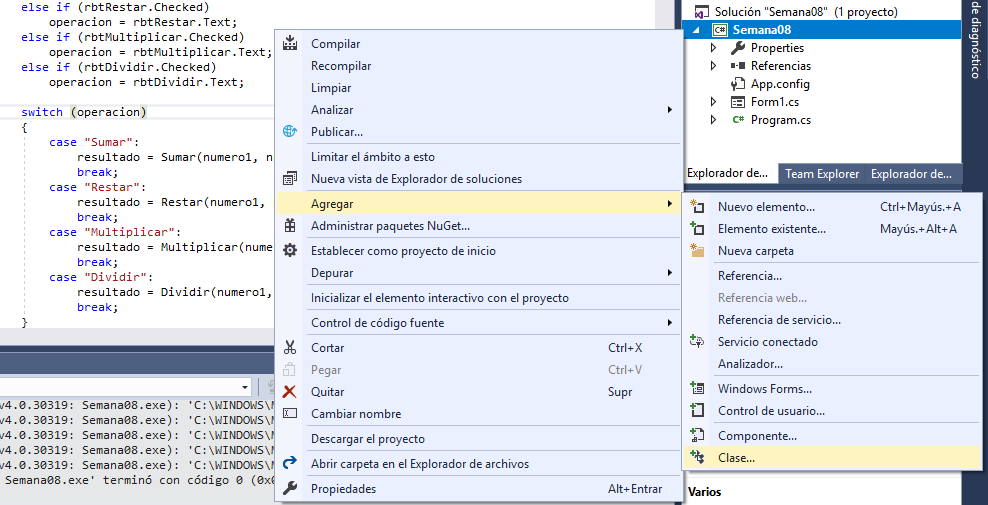
}

1. Ejecutamos la aplicación.

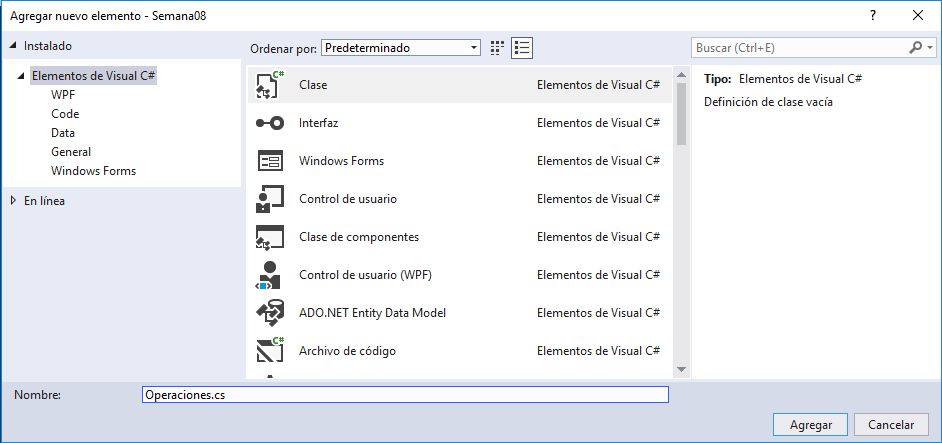


Paso 2. Clases y Métodos Estáticos.

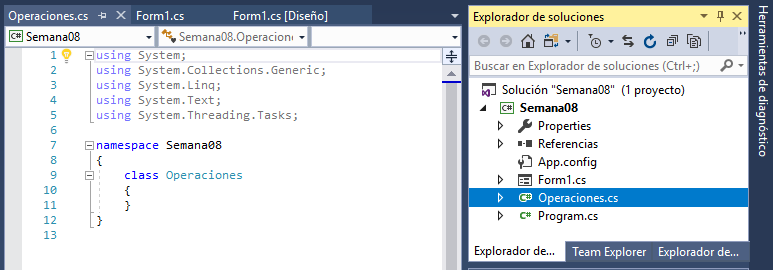
1. Vamos agregar un archivo cs donde ubicaremos nuestra clase estática. Hacemos click derecho en el proyecto y seleccionamos: Agregar > Clase…



1. Le ponemos el nombre “Operaciones.cs” y lo agregamos.



**Obtenemos:**



1. Cambiamos la definición de la clase de acuerdo a:

public static class Operaciones

{

}

1. Agregamos las mismas funciones que el ejemplo anterior pero con los modificadores public y static

public static class Operaciones

{

public static double Sumar(double pNumero1, double pNumero2)

{

double resultado = pNumero1 + pNumero2;

return resultado;

}

public static double Restar(double pNumero1, double pNumero2)

{

double resultado = pNumero1 - pNumero2;

return resultado;

}

public static double Multiplicar(double pNumero1, double pNumero2)

{

double resultado = pNumero1 \* pNumero2;

return resultado;

}

public static double Dividir(double pNumero1, double pNumero2)

{

double resultado = pNumero1 / pNumero2;

return resultado;

}

}

1. Guardamos los cambios.
2. En el código del formulario quitamos las definiciones de las cuatro operaciones.
3. Modificamos el código del evento Clic del botón de acuerdo a:

private void btnProcesar\_Click(object sender, EventArgs e)

{

double numero1 = Convert.ToDouble(txtNumero1.Text);

double numero2 = Convert.ToDouble(txtNumero2.Text);

double resultado = 0;

string operacion = "";

if (rbtSumar.Checked)

operacion = rbtSumar.Text;

else if (rbtRestar.Checked)

operacion = rbtRestar.Text;

else if (rbtMultiplicar.Checked)

operacion = rbtMultiplicar.Text;

else if (rbtDividir.Checked)

operacion = rbtDividir.Text;

switch (operacion)

{

case "Sumar":

resultado = Operaciones.Sumar(numero1, numero2);

break;

case "Restar":

resultado = Operaciones.Restar(numero1, numero2);

break;

case "Multiplicar":

resultado = Operaciones.Multiplicar(numero1, numero2);

break;

case "Dividir":

resultado = Operaciones.Dividir(numero1, numero2);

break;

}

lblResultado.Text = "Resultado: " + resultado.ToString();

}

1. Ejecutamos la aplicación y verificamos que tiene la misma funcionalidad.

**ArrayList.**

**Objetivos**

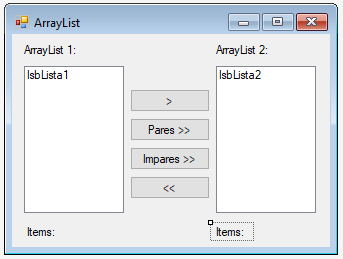
* Implementar aplicaciones con el uso de ArrayList.
* Aplicar la estructura iterativa Foreach para el recorrido de un ArrayList.

**Introducción**

Durante esta actividad, vamos aprender a utilizar colecciones de datos con ArrayList y gestionar su contenido.

Paso 1. Métodos de Clase.

1. Creamos un nuevo formulario y lo diseñamos de acuerdo a:



|  |  |  |
| --- | --- | --- |
| Control | Propiedad | Valor |
| Form2 | (Name) | frmEjemploArrayList |
|  | Text | ArrayList |
| ListBox1 | (Name) | lsbLista1 |
| ListBox2 | (Name) | lsbLista2 |
| Button1 | (Name) | btnPasarItem |
|  | Text | > |
| Button2 | (Name) | btnPasarPares |
|  | Text | Pares >> |
| Button3 | (Name) | btnPasarImpares |
|  | Text | Impares >> |
| Button4 | (Name) | btnRetornarTodo |
|  | Text | << |
| Label3 | (Name) | lblItemsLista1 |
|  | Text | Items: |
| Label4 | (Name) | lblItemsLista2 |
|  | Text | Items: |

1. Agregamos los ArrayList con los que trabajaremos en el ejemplo como atributos de la clase:

public partial class frmEjemploArrayList : Form

{

private ArrayList lista1 = new ArrayList();

private ArrayList lista2 = new ArrayList();

private int indiceSeleccionado = -1;

public frmEjemploArrayList()

{

InitializeComponent();

}

}

1. Creamos dos métodos que nos permitirán visualizar en los ListBox el contenido de los ArrayList:

private void mostrarLista1()

{

lsbLista1.Items.Clear();

foreach (int item in lista1)

{

lsbLista1.Items.Add(item.ToString());

}

lblItemsLista1.Text = "Items: " + lista1.Count;

}

private void mostrarLista2()

{

lsbLista2.Items.Clear();

foreach (int item in lista2)

{

lsbLista2.Items.Add(item.ToString());

}

lblItemsLista2.Text = "Items: " + lista2.Count;

}

1. Al mostrar el formulario vamos agregar contenido al primer ArrayList y mostrar su contenido en el primer ListBox:

private void frmEjemploArrayList\_Load(object sender, EventArgs e)

{

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

{

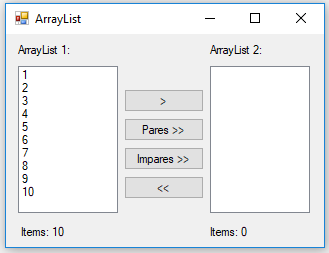
lista1.Add(i + 1);

}

mostrarLista1();

mostrarLista2();

}



Vamos agregar código al primer botón “>”, cuya función será de trasladar el ítem seleccionado desde el ArrayList 1 al ArrayList 2.

private void btnPasarItem\_Click(object sender, EventArgs e)

{

if(indiceSeleccionado > -1)

{

lista2.Add(lista1[indiceSeleccionado]);

lista1.RemoveAt(indiceSeleccionado);

}

indiceSeleccionado = -1;

mostrarLista1();

mostrarLista2();

}

1. Previamente debemos conocer el índice del ítem que ha sido seleccionado

private void lsbLista1\_SelectedIndexChanged(object sender, EventArgs e)

{

indiceSeleccionado = lsbLista1.SelectedIndex;

}

1. Vamos agregar código al segundo botón “Pares >>”, cuya función será de trasladar todos los números pares desde el ArrayList 1 al ArrayList 2.

private void btnPasarPares\_Click(object sender, EventArgs e)

{

int indice = 0;

foreach (int item in lista1)

{

if(item % 2 == 0)

{

lista2.Add(lista1[indice]);

}

indice++;

}

indice = 0;

while (indice < lista1.Count)

{

if (Convert.ToInt32(lista1[indice]) % 2 == 0)

{

lista1.RemoveAt(indice);

}

else

{

indice++;

}

}

mostrarLista1();

mostrarLista2();

}

1. Vamos agregar código al tercer botón “Impares >>”, cuya función será de trasladar todos los números impares desde el ArrayList 1 al ArrayList 2.

private void btnPasarImpares\_Click(object sender, EventArgs e)

{

int indice = 0;

foreach (int item in lista1)

{

if (item % 2 > 0)

{

lista2.Add(lista1[indice]);

}

indice++;

}

indice = 0;

while (indice < lista1.Count)

{

if (Convert.ToInt32(lista1[indice]) % 2 > 0)

{

lista1.RemoveAt(indice);

}

else

{

indice++;

}

}

mostrarLista1();

mostrarLista2();

}

1. Vamos agregar código al último botón “<<”, cuya función será de trasladar todos los items desde el ArrayList 2 al ArrayList 1.

private void btnRetornarTodo\_Click(object sender, EventArgs e)

{

foreach (int item in lista2)

{

lista1.Add(item);

}

while (lista2.Count > 0)

{

lista2.RemoveAt(0);

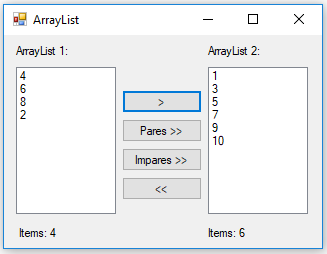
}

mostrarLista1();

mostrarLista2();

}

Ejecutamos la aplicación y verificamos su funcionalidad.



**Actividad:**

Ingresa a la plataforma virtual, revisa y analiza los siguientes enlaces, luego desarrollar las actividades propuestas:

* **Estructura if.**

<https://www.youtube.com/watch?v=SRT6uq5UTbc&list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B&index=3>

* **Estructura switch.**

<https://www.youtube.com/watch?v=p1xlu1uiO8&list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B&index=18>

* **Estructura For.**

<https://www.youtube.com/watch?v=Cx0UW5PQJG0&list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B&index=38>

* **Estructura While.**

https://www.youtube.com/watch?v=6gdFd9QXfTY&list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B&index=19

* **Estructura Do While.**

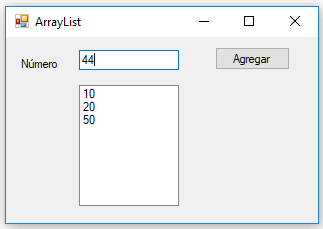
https://www.youtube.com/watch?v=yt05QtdO5Ek&list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B&index=21

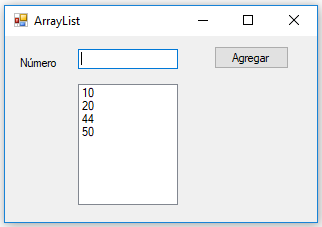
* **Métodos en C#.**
* <https://www.youtube.com/watch?v=FTJoQ71r3a4&list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B&index=9>
* <https://www.youtube.com/watch?v=N1OEk4gNFeQ&list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B&index=10>
* <https://www.youtube.com/watch?v=vAKn9ZOGnjQ&list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B&index=11>
* https://www.youtube.com/watch?v=CW00L7fk5gU&list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B&index=12
* **ArrayList.**
* <https://www.youtube.com/watch?v=GAal3fPN59g>
* https://www.youtube.com/watch?v=bdGSQ-9N22s

1. En un nuevo formulario, crea un ArrayList inicialmente vacío, desde un cuadro de texto se debe permitir agregar números enteros al ArrayList. Cada vez que se agrega un elemento al ArrayList:
   1. Debe colocarse de forma que la lista tenga un orden ascendente.
   2. Mostrar los elementos del ArrayList en un ListBox.

Debe usar la función Insert del ArrayList para agregar los números enteros.

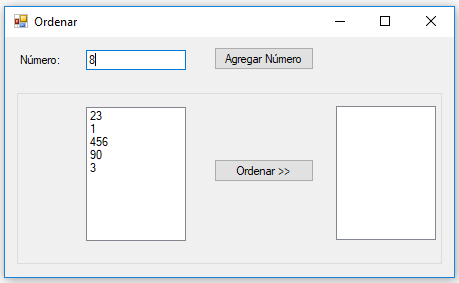
No debe modificar la propiedad Sorted del ListBox.





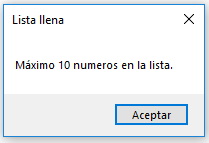
2.- En un nuevo formulario, implementar las siguientes funcionalidades:

a.- Diseño:



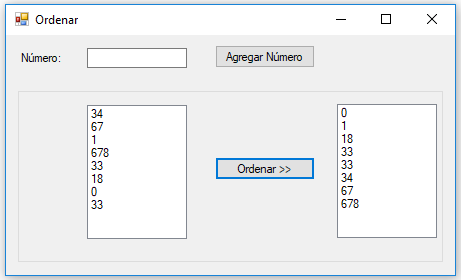
b.- Se debe llenar la primera lista con los números enteros que se ingresan en el cuadro de texto. Los números ingresados deben guardarse en un arreglo de dimensión 10.

* 1. La lista no debe superar los 10 ítems, de intentar ingresar más de 10 ítems debe mostrarse el siguiente mensaje:



* 1. Una vez que existan ítems en la primera lista, al presionarse el botón “Ordenar >>” debe ordenarse ascendentemente la lista de los números ingresados y mostrarse en la segunda lista.

Nota. No debe modificarse la propiedad “Sorted” de los ListBox.



* 1. Luego de ordenar la lista de números debe permitirse ingresar más números (sin superar los 10) y reordenarlos nuevamente.