|  |  |
| --- | --- |
|  | **UNIVERSIDAD ABIERTA INTERAMERICANA** |
|  | **FACULTAD DE TECNOLOGÍA INFORMÁTICA** |
| **Asignatura: Programación I Trabajo práctico obligatorio Nro 1** |
|  | **Fecha entrega:** 01 de Octubre 2020 |

|  |  |
| --- | --- |
| **Alumno:** Julián Emilio Ryan | **Legajo:** B00044969-T4 |
| **Obs:** | **Nota:** |

# Dentro del contexto del IDE de Visual Studio:

**Teoría**

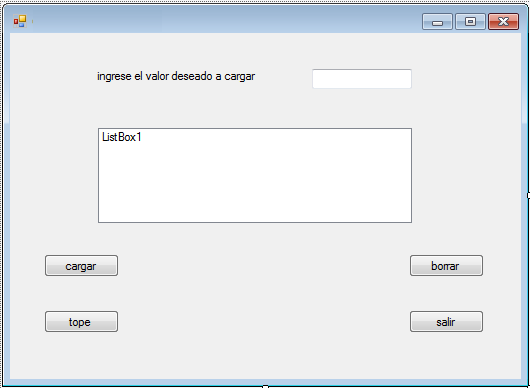
1. ¿Para qué se utiliza la ventana denominada Solution Explorer?
2. ¿Para qué se utiliza la ventana denominada Properties?
3. ¿Para qué se utiliza la ventana denominada Toolbox?

# Dentro del contexto del lenguaje C#

1. ¿Cómo se declara un vector de tipo int de 10 elementos?
2. ¿Qué ámbitos existen para asignarle a los atributos y a los métodos?
3. ¿Qué significa que el parámetro de un método se pasa por valor o referencia?
4. ¿Para qué se utiliza null?
5. Enumere las estructuras de decisión. Ejemplifique cada una de ellas y explique en qué se diferencian.
6. Enumere las estructuras de repetición. Ejemplifique cada una de ellas y explique en qué se diferencian.
7. ¿Cómo diseñaría una lista enlazada simple de una clase llamada Profesor con los atributos Nombre (String) y Legajo (int)? Grafique y explique.

**Práctica**

Dado el siguiente formulario:



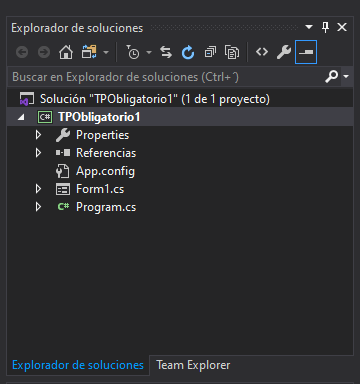
1. Implemente en el formulario una cola (FIFO) generando usted mismo la estructura interna de funcionamiento. Genere una clase contenedora llamada Alumno que tendrá como atributo principal el nombre que escribirá el usuario en el textbox del formulario (Puede agregar más atributos si así lo desea) y luego se agregará al listbox.
2. Agregue la posibilidad de cargar el Alumno desde un segundo formulario que permita retornar un objeto de tipo Alumno si el usuario selecciona *“*Aceptar*”* desde un botón asignado al mismo, también se deberá poder cancelar la operación.
3. El botón cargar, agrega un elemento a la cola y al listbox (Queue). El botón tope, muestra el primer elemento de la cola, pero no lo quita (Peek). El botón borrar, muestra el primer elemento agregado a la cola y lo quita (Dequeue).

Desarrollo:

Dentro del contexto del IDE de Visual Studio:

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

La ventana denominada Solution Explorer es una ventana especial que le permite administrar soluciones, proyectos y archivos. Proporciona una vista completa de los archivos de un proyecto y permite agregar o eliminar archivos y organizarlos en subcarpetas.

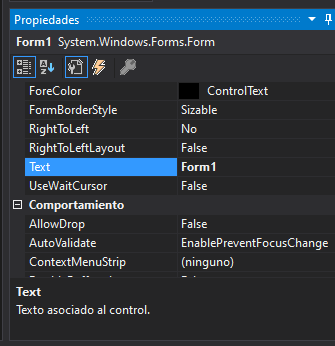


*Explorador de soluciones en Visual Studio*

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

La ventana denominada Properties se utiliza para ver y cambiar las propiedades en tiempo de diseño y los eventos de los objetos seleccionados que se encuentran en editores y diseñadores. También se puede utilizar para editar y ver propiedades de archivos, proyectos y soluciones.

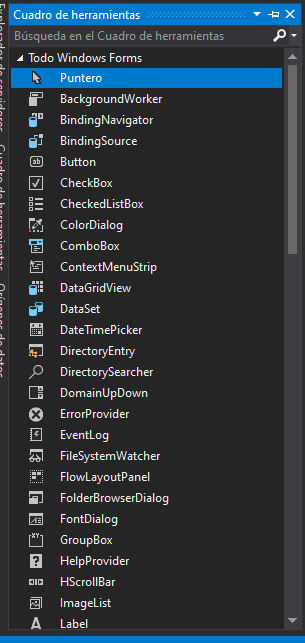
La ventana Propiedades muestra diferentes tipos de campos de edición, según las necesidades de una propiedad en particular. Estos campos de edición incluyen cuadros de edición, listas desplegables y enlaces a cuadros de diálogo de editor personalizados. Las propiedades que se muestran en gris son de solo lectura.



*Ventana de Propiedades en Visual Studio*

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

La ventana denominada Toolbox se utiliza para mostrar los controles que puede agregar a proyectos de Visual Studio. El usuario puede arrastrar y colocar distintos controles en la superficie del diseñador que use y cambiar su tamaño y posición.



*Ventana de Cuadro de Herramientas en Visual Studio*

Dentro del contexto del lenguaje C#

1. ¿Cómo se declara un vector de tipo int de 10 elementos?

Se declara colando el tipo de dato que va a alojar el vector, la cantidad de posiciones que va a tener y se le asigna un nombre.

Ejemplo:

int [10] precios;

1. ¿Qué ámbitos existen para asignarle a los atributos y a los métodos?

Todos los ámbitos tienen un nivel de accesibilidad. El nivel de accesibilidad controla si se pueden usar desde otro código del ensamblado u otros ensamblados. Los ámbitos que existen para asignar son:

* public: Puede obtener acceso al tipo o miembro cualquier otro código del mismo ensamblado o de otro ensamblado que haga referencia a éste.
* private: Solamente el código de la misma class o struct puede acceder al tipo o miembro.
* protected: Solamente el código de la misma class, o bien de una class derivada de esa class, puede acceder al tipo o miembro.
* internal: Puede obtener acceso al tipo o miembro cualquier código del mismo ensamblado, pero no de un ensamblado distinto.
* protected internal: Cualquier código del ensamblado en el que se ha declarado, o desde una class derivada de otro ensamblado, puede acceder al tipo o miembro.
* private protected: El código de la misma class, o de un tipo derivado de esa class, puede acceder al tipo o miembro solo dentro de su ensamblado de declaración.

Ejemplo:

public class Bicicleta

{

public void Pedal() { }

}

1. ¿Qué significa que el parámetro de un método se pasa por valor o referencia?

C# provee dos tipos de datos, por Valor y por Referencia. El proceso de conversión de tipos por Valor a tipos por Referencia es conocido como *boxing* y la conversión de tipos de Referencia a tipos por Valor es conocido como *unboxing*.

1. ¿Para qué se utiliza null?

null se utiliza tratar valores nulos. Como por ejemplo cuando iniciamos un primer nodo en una Lista.

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

- Estructura if: La estructura IF permite controlar la ejecución o no de una porción de código según se cumpla una función establecida. Si la condición es verdadera, entonces la instrucción se ejecuta; en este caso, «condición» debe ser una expresión que, una vez evaluada, debe devolver una booleana true o false. Con esta sintaxis, sólo la instrucción colocada después del if, se ejecutará si la condición es verdadera.

Ejemplo:

if (condición) {

condición verdadera;

} else {

condición falsa;

}

- Estructura switch: La estructura SWITCH permite un funcionamiento equivalente a la estructura IF, pero con más posibilidades de ejecución de código según el valor de una variable. El valor de la variable se evalúa al principio de la estructura (por el switch). Luego el valor obtenido se compara con el valor especificado en el primer case (valor1). Si los dos valores son iguales, entonces el bloque de código 1 se ejecuta. Si no, el valor obtenido se compara con el valor del case siguiente; si hay correspondencia, el bloque de código se ejecuta y así sucesivamente hasta el último case. Si ningún valor concordante se encuentra en los diferentes case, entonces el bloque de código especificado en el default se ejecuta. Cada uno de los bloques debe terminarse con la instrucción break. El valor que hay que probar puede estar contenido en una variable, pero también puede ser el resultado de un cálculo.

Ejemplo:

switch (variable)

{

case valor1:

Bloque de código 1

case valor2:

Bloque de código 2

case valor3:

Bloque de código 3

default:

Bloque de código 4

}

En este caso, el cálculo sólo se efectúa una vez al principio del switch. El tipo del valor probado puede ser numérico o cadena de caracteres. El tipo de la variable probada debe corresponder por supuesto al tipo de los valores en los diferentes case.

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

- For: Esta estructura nos permite la posibilidad de generar un bucle que se repita una cantidad determinada de veces. En la cabecera se indica la variable de control, la condición de repetición y el incremento o decremento de la misma.

Ejemplo:

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

Bloque de código

}

- While: Este tipo de iteración, se diferencia porque seguirá ejecutándose hasta que una condición determinada deje de cumplirse y no cuando se cumpla. Dentro de este tipo de bucles se pueden usar condiciones numéricas, pero también de tipo booleano, carácter o cadena de texto. La ejecución del ciclo while, no se realiza si la condición no se cumple al momento de ejecutar al código.

Ejemplo:

int i = 0;

while (i < 5) {

Console.WriteLine(i);

i++;

}

Ejemplo:

bool entra\_bucle = false;

while(entra\_bucle == true){

Bloque de código

}

- Do while: Esta estructura, es similar a la previa, con la particularidad de que la condición se comprobará al final de ejecución del código y no al principio, a diferencia de como ocurría tanto con los bucles For y While. El código se ejecutará al menos una vez, aunque la condición sea falsa. Resulta útil cuando se desea que la ejecución del código ocurra al menos una vez o cuando queremos que la condición de nuestra estructura de iteración se verifique al final del bloque de código.

Ejemplo:

do{

Console.WriteLine(i);

i++;

} while (i < 5)

- For each: Este tipo de bucle permite recorrer colecciones o arreglos completamente. No existe ningún tipo de condición, excepto que debemos completar toda la colección que hayamos querido ver. Este tipo de bucles, resulta útil para evitar las excepciones del tipo fuera del índice.

Ejemplo:

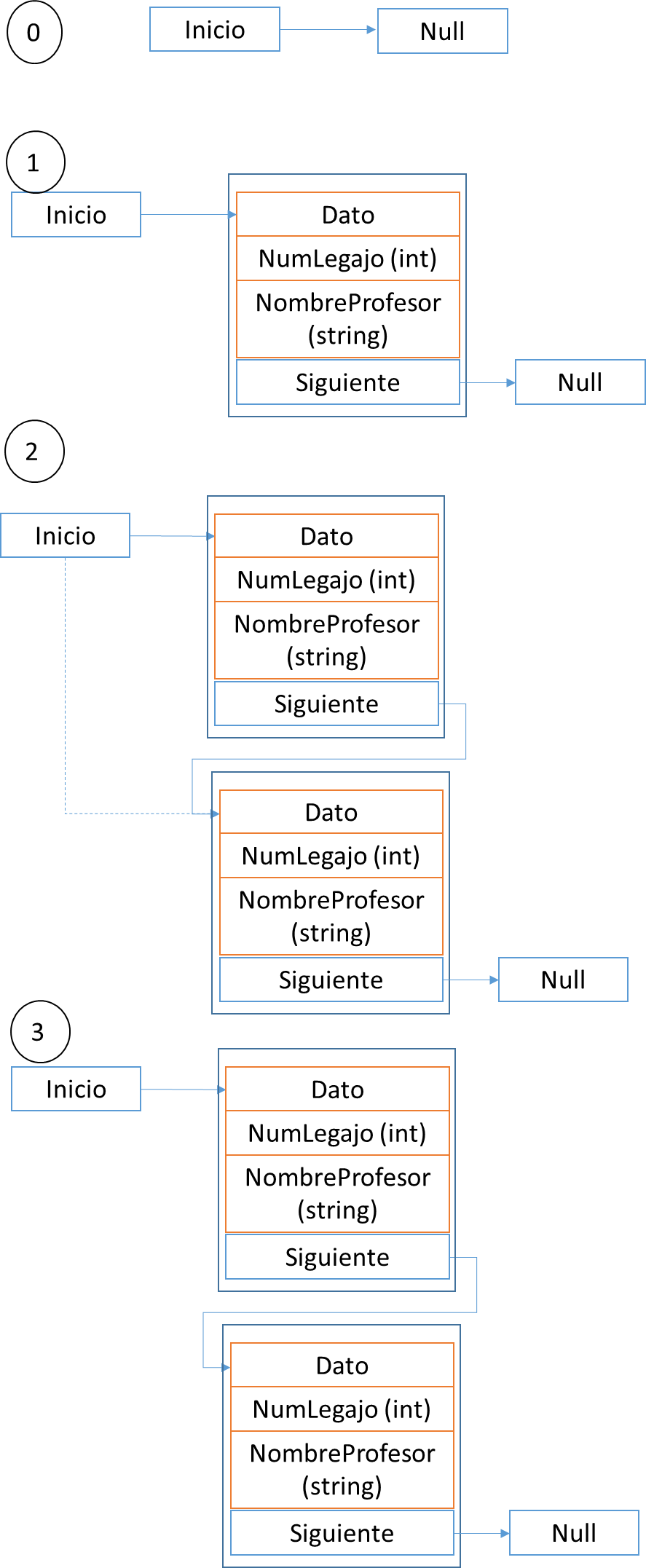
string[] arreglo = new string[5];

Foreach(string cad in arreglo){

Bloque de código

}

1. ¿Cómo diseñaría una lista enlazada simple de una clase llamada Profesor con los atributos Nombre (String) y Legajo (int)? Grafique y explique.



Comenzaría estableciendo las clases para Profesor con los atributos NumLegajo (int) y NombreProfesor (String), a continuación, establecería la clase con la estructura del nodo donde este contenga los atributos Datos y Siguiente.

Después, daría inicio a un Nodo inicial con valor nulo, luego cargaría en la parte de Siguiente del Nodo Inicial un Nuevo Nodo donde pediría los datos al usuario, estos datos serían los primeros ingresados y quedarían en al principio de nuestra Lista simple. Seguiría apilando de esta manera los nodos. El Primer Nodo con datos apunta al Nodo con valor nulo, y en la siguiente repetición del programa se creará un Segundo Nodo con datos que cuyo puntero siguiente apunte al Primer Nodo con datos, ingresando siempre los datos desde Nuevos Nodos.

Parte Práctica

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TPObligatorioN1

{

class Program

{

static void Main(string[] args)

{

Cola ColaAlumnos = new Cola();

int opc = 0;

Console.WriteLine("\*\*Software de Cargado de Colas de Alumnos\*\*\n\n");

do

{

Console.WriteLine("\n\n Ingrese la accion que desea realizar: ");

Console.WriteLine("1) Cargar alumnos en la Cola de Alumnos");

Console.WriteLine("2) Mostrar la Cola de Alumnos");

Console.WriteLine("3) Mostrar al primero alumno de la Cola de Alumnos");

Console.WriteLine("4) Borrar el primer alumno agregado a la Cola de Alumnos");

Console.WriteLine("Ingrese la accion que desea realizar (cero para salir): ");

opc = Int32.Parse(Console.ReadLine());

switch (opc)

{

case 1:

ColaAlumnos.insertarNodo();

break;

case 2:

ColaAlumnos.mostrarCola();

break;

case 3:

ColaAlumnos.mostrarPrimero();

break;

case 4:

ColaAlumnos.borrarPrimero();

break;

default:

Console.WriteLine("Ingreso un valor incorrecto, vuelva a intentar");

break;

}

} while (opc != 0);

Console.WriteLine("A seleccionado salir del Software de Cargado de Colas de Alumnos. Hasta luego!\n\n");

Console.ReadLine();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TPObligatorioN1

{

class Cola

{

private Nodo Primero = new Nodo();

private Nodo Ultimo = new Nodo();

public Cola()

{

Primero = null;

Ultimo = null;

}

public void insertarNodo()

{

Nodo Nuevo = new Nodo();

//Nombre = textBox1.Text;

Console.WriteLine("Ingrese el nombre del Alumno: ");

Nuevo.Nombre = Console.ReadLine();

if (Primero == null)

{

Primero = Nuevo;

Primero.Siguiente = null;

Ultimo = Nuevo;

}

else

{

Ultimo.Siguiente = Nuevo;

Nuevo.Siguiente = null;

Ultimo = Nuevo;

}

}

public void mostrarCola()

{

Nodo Actual = new Nodo();

Actual = Primero;

if (Primero != null)

{

while (Actual != null)

{

Console.WriteLine("El nombre del alumno/a es " + Actual.Nombre);

Actual = Actual.Siguiente;

}

}

else

{

Console.WriteLine("\n La Cola de Alumnos se encuentra vacia \n");

}

}

public void mostrarPrimero()

{

//Nodo Actual = new Nodo();

if (Primero != null)

{

Console.WriteLine("El nombre del alumno/a es " + Primero.Nombre);

}

else

{

Console.WriteLine("\n La Cola de Alumnos se encuentra vacia \n");

}

}

public void borrarPrimero()

{

if (Primero != null)

{

Console.WriteLine("Se elimino el nombre del alumno/a " + Primero.Nombre);

Primero = Primero.Siguiente;

}

else

{

Console.WriteLine("\n La Cola de Alumnos se encuentra vacia \n");

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TPObligatorioN1

{

class Nodo

{

public string Nombre { get; set; }

public Nodo Siguiente { get; set; }

}

}