

CTS Define un conjunto de tipos de datos orientados a objetos, todo tipo hereda directa o indirectamente de System.Object y CTS define tipos de valor y referencia.

Valor: Int, char, byte, char, double, enum

Referencia: Class, String, Array, Interface

Tipos de Datos: c#

Enteros:

Byte	byte
------	------

SByte	sbyte
-------	-------

Int16	short
-------	-------

Int32	int
-------	-----

Int64	long
-------	------

Punto Flotante

Single	float
--------	-------

Double	double
--------	--------

Decimal	decimal
---------	---------

Logicos

Boolean	bool
---------	------

Otros

Char	char
------	------

Object	object
--------	--------

String	string
--------	--------

Valores predeterminados de atributos de clase :

- Enteros y pto. Flotante = 0
- Logicos = False
- Referencias = null

Tipo de Conversiones:

- Explicitas: Interviene el programador porque puede haber perdida de datos.(casteo)
- Implicita: No interviene el programador

Ej: Un int puedo convertirlo a long, float, double, decimal

Un float puedo convertirlo en double, decimal

Un sbyte puede ser, short, int, long, double, float, decimal

Punto de entrada:

Static void Main(string[] args)

Clase Console: public static class Console

Representa la entrada, salida y errores de streams es miembro de Namespace System.

Metodos:

- .clear() limpia pantalla
- .Read() lee proximo carácter de stream y devuelve un entero
- .ReadKey(bool) obtiene el carácter presionado por el usuario, si es true lo muestra.
- .ReadLine() lee los caracteres hasta presionar enter devuelve string
- .Write() imprime por pantalla el string que se pasa como parámetro
- .WriteLine() imprime un string pasado y produce un salto de línea.

Propiedades:

- .BackColor() Obtiene o establece color de fondo de consola.
- .ForeColor() establece u obtiene color de texto de consola.
- .Title es el titulo de la ventana de consola.

Formatos:

{ N [, M][: Formato] }

N es el numero de parámetro

M es el ancho usado para mostrar el parámetro

Formato es una cadena que indica el formato extra a usar con ese parámetro.

POO es un nuevo paradigma, propone resolver problemas de la realidad a través de identificar objetos y relaciones de colaboración entre ellos

Pilares Principales:

- Herencia:
 - Es un tipo de relación entre clases, va de la generalización a la especialización, hereda la implementación. Una clase comparte la estructura y el comportamiento definido en otra clase.
 - Cada clase que hereda de otra posee:
 - Atributos de clase base y los propios
 - Soporta todos o alguno de los métodos de la clase base
 - El proposito es organizar mejor las clases que componen una determinada realidad y poder agruparlas en función de atributos y comportamientos comunes.
 - Simple: Clase derivada hereda de la base
 - Multiple: la derivada hereda de multiples padres
- Polimorfismo
 - La definicion el método reside en la clase base, la implementación del método reside en la clase derivada, la invocación es resuelta al momento de la ejecución.
- Encapsulamiento
- Abstraccion

CLASES

Es una clasificación, clasificamos en base a comportamientos y atributos comunes. Es una abstracción de un objeto.

Es una construcción estática que describe un comportamiento común y atributos(estado)

Es una estructura de datos.

Incluye datos y métodos.

[modificador] class Identificador (Sustantivos y con primer letra Mayuscula)

{// miembros: atributos y métodos }

Modificadores:

- **Abstract** la clase no podrá instanciarse
- **Internal** Accesable a todo el proyecto
- **Public** Accesible desde cualquier proyecto
- **Private** acceso por defecto
- **Sealed** la clase no se puede heredar

ATRIBUTOS

[modificador] tipo identificador; // Igual que en C (LETRAS EN MINUSCULAS, SI ES PRIVADO O PROTEGIDO PRIMERO _)

Modificadores:

- **Private** los miembros de la misma clase acceden.
- **Protected** los miembros de la misma clase o clases derivadas acceden.
- **Internal** los miembros del mismo proyecto
- **Internal protected** los miembros del mismo proyecto o clase derivada.
- **Public** cualquier miembro.

METODOS

[modificador] retorno Identificador ([args]) (Verbos, primer letra con mayuscula
{ // Sentencias }

Los parámetros se definen Tipo nombre y se separan por coma.

Modificadores:

- **Abstract** Solo firma del método
- **Extern** firma del método
- **Internal** accesible desde el mismo proyecto
- **Override** Reemplaza la implementación del mismo método declarado como virtual en una clase padre.
- **Public** accesible desde cualquier proyecto
- **Private** Accesible solo desde la clase
- **Protected** Solo accesible desde clase o derivadas
- **Static** indica que es un método de clase
- **Virtual** Permite definir métodos, con su implementación que podrán ser sobreescritos en clases derivadas.

Ejemplo

```
public class Automovil
{
    // Atributos NO estáticos
    public Single velocidadActual;
    // Atributos estáticos
    public static Byte cantidadRuedas;
    // Métodos estáticos
    public static void MostrarCantidadRuedas()
    {
        Console.Write(Automovil.cantidadRuedas);
    }
    // Métodos NO estáticos
    public void Acelerar (Single velocidad)
    {
        this.velocidadActual += velocidad;
    }
}
```

NAMESPACE

Es una agrupación lógica de clases y otros elementos, toda clase está dentro de un namespace. Su función principal es la organización del código para reducir los conflictos entre nombres.

Ej System.Console.WriteLine()
System es el NameSpace
Console la clase
WriteLine el método

Directivas: Permiten el uso de los miembros sin tener que especificar un nombre completo.

- Using : Permite la especificación de una llamada a un método sin el uso obligatorio de un nombre completamente cualificado. Using system
- Alias: Permite que un programa utilice un nombre distinto para un NameSpace, Es una técnica usada para conseguir una notación abreviada que evite el uso de NameSpaces largos using SC = System.Console;

```
namespace Identificador
{
    Clases, enumerados, namespaces, using, alias, interfaces }
```

OBJETOS

Son instancias de una clase, son creados en tiempo de ejecución, poseen comportamientos(métodos) y estado(atributos), para acceder a los métodos y atributos se utiliza el .. Para crear un objeto se utiliza new.

```
Nombre_clase nombre_objeto = new Nombre_Clase();
Ident. Clase    nombre                constructor del objeto
```

Tiempo de vida de objeto

- Se crea el objeto
- Se inicializa con constructor
- Se utilizan métodos y atributos
- Se vuelve a convertir el objeto en memoria
- Se libera la memoria

El tiempo de vida de una variable local está vinculado al ámbito en el que esta declarada, el tiempo de vida de un objeto dinámico no está vinculado a su ámbito.

- El stack es liberado automáticamente (variables locales) en cambio el Heap está controlado por el GarbageCollector

Los objetos se crean pero no se destruye al final del ámbito en donde son creados, no es posible controlar cuando se destruye un objeto.

No es posible destruir objetos de forma explícita.

El GC actúa cuando hay poca memoria o el programa está por finalizar

CONSTRUCTORES

Son métodos especiales que se utilizan para inicializar objetos al momento de su creación.

Aunque no exista un constructor, existe uno por defecto.

Los constructores llevan el mismo nombre que la clase.

Tipos:

- De instancia: inicializan objetos
- Estáticos : son los que inicializan clases.

Constructor por defecto:

No reciben parámetros, tiene el mismo nombre que la clase, inicializa los campos en cero o null o false.

```
class MiClase
{
    public MiClase()
    {
        //Inicializar atributos de instancia aquí
    }
}
```

Los constructores estáticos son encargados de inicializar clases, solo inician los atributos estáticos, no llevan modificadores de acceso, utilizan static, no pueden recibir parámetros

```
class Clase {
    private static int _var1;
    private static int _var2;
    private int _var3;
    static Clase() {
        _var1 = 3;
        _var2 = 5;
    }
}
class Test {
    static void Main() {
        new Clase();
    }
}
```

SOBRECARGAS

METODOS Y CONSTRUCTORES

- Se deben cambiar el numero, tipo y orden de los parámetros, se distinguen métodos sobrecargados comparando la lista de parámetros.

Firma:

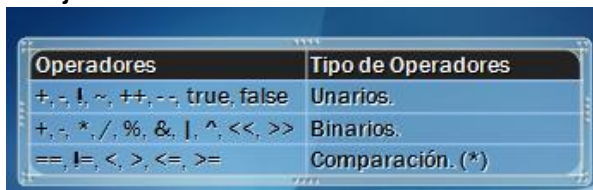
- Nombre
- Tipo de parámetros
- Cantidad de parámetros
- Modificador out o ref
- No afectan la firma:
 - + Nombres de parámetros
 - + tipo de retorno

OPERADORES

Consiste en modificar el comportamiento cuando este se utiliza con una determinada clase:

[acceso] static TipoRetorno operator nombreOperador (Tipo a[, Tipo b])

{ .. }



Operadores	Tipo de Operadores
+, -, !, ~, ++, --, true, false	Unarios.
+, -, *, /, %, &, , ^, <<, >>	Binarios.
==, !=, <, >, <=, >=	Comparación. (*)

Convesiones definidas: Permien hace compatibles dos elementos que no lo eran, pueden haber operadores implícitos o explícitos

public static implicit operator nombreTipo (Tipo a)

public static explicit operator nombreTipo (Tipo a) (Tipo)

HERENCIA

[modificadores] class NombreClase : NombreClaseBase

- Una clase derivada hereda todo menos los constructores base
- Miembros públicos de base son públicos en la derivada
- Los miembros de clase tienen acceso a miembros privados de esta clase, aunque la clase derivada también los hereda.
- Una clase derivada no puede ser más accesible que su clase base
 - Protected: el acceso depende de la relación entre la clase que tiene el modif y la clase que intenta acceder a los miembros que usan el modif.
 - Para una clase privada es equivalente a public
 - Entre dos clases sin relación se comportan como privados.
 - Clase derivada hereda el miembro protected
 - Las clases que deriven directa o indir. Pueden acceder a los miembros protegidos.
 - Los métodos de una clase derivada solo tienen acceso a los propios miembros heredados con protected

```

class Clase : ClaseDerivada    }
{
void Compila()
{
    Console.WriteLine(_edad);
}
}

```

Constructores de clase base: [modificadores] Constructor([ListaArgs]) : base([ListaArgs])

- Una clase sin clases base explícitas extiende implícitamente la clase **System.Object**, que contiene un constructor público sin parámetros.
- el compilador generará un mensaje de error si el constructor indicado no coincide con ningún constructor de la clase base

WINDOWS FORMS System.Windows.Forms

- Formulario : Los formularios son objetos que exponen propiedades, métodos que definen su comportamiento y eventos que definen la interacción con el usuario.
- Objetos:
 - o Size
 - o Location
 - o Font

.show() Visualiza el formulario

.close() cierra formulario

.hide() oculta el formulario

Ciclo de vida:

- **New:** Se crea la instancia del formulario. (*)
- **Load:** El formulario está en memoria, pero invisible.
- **Paint:** Se “pinta” el formulario y sus controles.
- **Activated:** El formulario recibe foco.
- **FormClosing:** Permite cancelar el cierre.
- **FormClosed:** El formulario es invisible.
- **Disposed:** El objeto está siendo destruido.
- Message Box: Para mostrar información o pedir intervención del usuario. Esta clase contiene métodos estáticos que permiten mostrar un cuadro de mensaje para interactuar con el usuario de la aplicación.
 - o MessageBoxButtons.AbortRetryIgnore
 - o MessageBoxIcon.Error
 - o MessageBoxDefaultButton.Button1

Constroles:

- Propiedades:
 - o Name: indica el nombre utilizado en el código para identificar el objeto.
 - o BackColor es el colo de fondo del componente
 - o Cursor es el cursor que aparece al pasar el puntero por el control
 - o Enabled indica s el control esta habilitado.
 - o Font es la fuente utilizada para mostrar el texto
 - o ForeColor color utilizado en el texto.
 - o Locked determina si se puede o no cambar el tamaño del control
 - o Modifiers indica el nivel de visibilidad del objeto.
 - o TabIndex detrmina el índice del orden de tabulación que ocpara el control.

- Text es el texto asociado al control.
- Visible determina si el control esta visible o no.
 - CheckBox: casilla de verificación que podemos marcar para indicar un estado(V o F o casilla color gris)
 - RadioButton : Permiten definir conjuntos de opciones auto excluyentes.
 - GroupBox: Permite agrupar controles en su interior
 - ListBox: Contiene una lista de valores de los cuales se puede seleccionar uno o varios simultaneamente.
 - ComboBox: Es un control basado en la combinación de TextBox y ListBox, dispone de una zona de edición de texto y una lista de valores.

Clase 7

Arrays: puede ser unidimensional, multidimensional o anidado, el valor por defecto de elementos numéricos es cero, en objetos se establece null.

- Son base cero
- Pueden ser de cualquier clase.
- Son reference type, heredan de system.array
- Implementan interface IEnumerable, son iterables con foreach

Syntaxis:

[acceso] Tipo [] nombre_Array = new Tipo[CANT_ELEMENTOS];

Antes de utilizarlo hay que instanciarlo

Metodos

- CopyTo : copia los elementos del array actual a uno especificado desde un índice establecido.
- GetLength: obtiene la cantidad de elementos para una dimension determinada del array.
- GetLowerBound/GetUpperBound : obtiene limite superior o inferior para una dimension del array.
- GetValue/SetValue : obtiene o establece el valor de uno o varios elementos del array.
- Initialize inicializa elemento del array mediante constructor.

Propiedades

- Length: obtiene la cantidad total de elementos de todas las dimensiones del array
- Rank obtiene el numero de dimensiones del array

Multidimensional

[acceso] Tipo [,] nombre_Array = new Tipo[FILAS, COLUMNAS];

MANEJO DE CADENAS System.String

Metodos Estaticos:

- Compare retorna 1 0 -1
- Concat
- Copy

Metodos de instancia:

- CompareTo idem compare
- Contains : indica si una cadena esta o no en el objeto actual
- CopyTo idem copy()
- EndsWith/StartWith: determina si el final o principio de la cadena contenida por el objeto coincide o no con la cadena pasada como argumento.
- IndexOf/LastIndexOf devuelve el índice de la primera/ultima ocurrencia de la cadena por parámetro.
- Insert : inserta una cadena a partir de una posición determinada
- Remove: borra todos los caracteres a partir de una posición
- Replace: reemplaza todas las ocurrencias de una cadena especificada por otra especificada en la instancia por otra cadena.
- Substring retorna una subcadena a partir de un índice y un largo.
- ToLower/ToUpper devuelve copia en mayúscula o minúscula
- TrimEnd/TrimStart: Remueve todas las ocurrencias de un conjunto de caracteres especificado en un Array, desde el final/principio de la instancia.
- Trim : Remueve todos los espacios en blanco del principio y del final de la instancia.

Propiedades:

- Chars: Obtiene el carácter situado en una posición de carácter especificada en el objeto System.String actual
- Length: Retorna la cantidad de caracteres que posee la instancia.

COLECCIONES:

- No Genericas:

- Pilas(Stack) Ultimo en entrar, primero en salir
 - Metodos:
 - Clear: Remueve todos los elementos de la pila
 - Peek: retorna el elemento de la cima sin removerlo
 - Pop : retorna elemento y lo elimina
 - Push : Inserta un objeto en la cima
 - ToArray : copia el contenido de la pila en un array tipo object.
 - Propiedades
 - Count devuelve el numero de elementos de la pila.
- Colas:
 - Metodos:
 - Clear : Remueve elementos de la cola
 - Dequeue : Retorna y remueve el primer elemento de la cola
 - Enqueue : inserta objeto al final de la cola
 - Peek: retorna primer elemento sin removerlo
 - ToArray copia el contenido en un Array de tipo Object.
 - Propiedades:
 - Count : Devuelve numero de elementos contenidos.

- Listas : Colección Arraylist, puede cambiar su tamaño dinámicamente
 - Metodos:
 - Add: Agrega un objeto al final.
 - Clear : remueve todos los elementos.
 - Insert: inserta un elemento en la posición indicada
 - Remove : remueve el elemento seleccionado.
 - Reverse : Invierte el orden de toda la colección
 - Sort: Ordena el contenido de la colección
 - Propiedades:
 - Capacity : Obtiene o determina el nro de elem
 - Count : Devuelve el nro de elementos contenidos.
- Hastable: Operan sobre el principio de pares clave/valor. Cada valor debe añadirse y recuperarse por la misma clave.
 - Metodos:
 - Add: Agrega un elemento con una clave esp.
 - Clear : Remueve todos los elementos
 - Remove: Remueve un objeto con una clave esp.
 - Propiedades:
 - Count: Devuelve el numero de pares contenidos en a colec.

Listas Genericas: Usado para crear clases de colecciones. Aumenta la seguridad de tpos de datos, aumenta la reutilización de código y performance (unboxing, boxing), se limita el acceso a determinados métodos según el tipo de datos utilizado en la clase Generic. Se encuentra en System.Collections.Generic

- Dictionary<Tclave,Tvalor> : representa una colección clave, valor
- List<T>:Representa una lista de objetos fuertemente tipado, que puede ser accedida por índice. Provee métodos para buscar, ordenar y manipular listas.
- Stack<T>: Representa una colección de objetos genéricos del tipo LIFO (Last In First Out).
- Queue<T> : Representa una colección de objetos genéricos del tipo FIFO (First In First Out).

Gestion de Errores:

Es una tecnica que permite interceptar los errores en tiempo de ejecucion ya sean esperados o no. Cuando este se produce, se lanza una excepcion.

- Todas las excepciones derivan de la clase Exception
- Cada clase de excepcion puede residir dentro de su propio archivo de origen sin tener relaciones con otras clases.
- Cada clase de excepcion contiene informacion especifica sobre el error.
- Cada clase de excepcion es descriptiva y representa un error concreto de forma clara y evidente.

Bloques try catch

Separa fisicamente las instrucciones basicas del programa, el flujo de control normal y por otra parte el tratamiento de excepciones.

Las partes de código que pueden llegar a lanzar excepciones se colocan dentro del try, mientras el tratamiento de excepciones en el bloque try se ponen en el catch.

- El bloque contiene una expresión que puede generar la excepción, en caso de producirse el runtime detiene la ejecución normal y comienza a buscar un bloque catch que pueda capturar la excepción basándose en su tipo.
- Si la función inmediata no se encuentra dentro de un bloque catch, el runtime busca el bloque catch que capture la excepción, de lo contrario se cerrará el programa.

Un bloque de código puede producir una o más clases de excepción, al haber muchas clases, es posible que haya muchos catch y que cada uno capture una exp diferente.

En caso que exista un bloque catch general, este debe ser el último en la lista.

Se lanzan excepciones con THROW, el runtime detiene la ejecución normal y transfiere la exc al bloque catch más cercano, se pueden lanzar excepciones propias también.

Bloque finally

- Contiene conjunto de instrucciones a ejecutar sea cual sea el flujo de control, aunque abandone el bloque try.
- También se ejecutará si abandona un try por un throw o por un break o continue.
- Es útil para evitar la repetición de instrucciones y para liberar recursos tras el lanzamiento de la exc.

LECTURA/ESCRITURA DE ARCHIVOS

- La clase StreamWriter escribe caracteres en archivos de texto, StreamReader los lee.
- Ambas clases se encuentran en SYSTEM.IO
 - o StreamWriter(path) inicia una nueva instancia, si el archivo existe se sobrescribirá, sino se crea./ idem reader

- StreamWriter(path,bool) Si es true hace append si el archivo existe
- StreamWriter(path,bool,encoding) se especifica tipo de codificación/idem read
 - Instancia.Write(string) escribe sin provocar salto de línea
 - Instancia.WriteLine(string)
 - Instancia.close()
- StreamReader
 - Var = instancia.Read() Lee carácter a carácter y retorna un entero
 - Var = instancia.ReadLine() Lee línea hasta salto de línea.
 - Var = instancia.ReadToEnd() lee todo el stream y lo devuelve como cadena.

SERIALIZACION

- Es el proceso de convertir un objeto de memoria en una secuencia lineal de bytes.
- Se utiliza para pasar datos a otro proceso, maquina, grabarlo en base de datos.
- Serializacion a xml
 - Incluyen propiedades y atributos públicos
- Serializacion Binaria
 - Incluyen propiedades y atributos privados o públicos.
 - No serializa propiedades, indexadores, atributos de solo lectura.
- La clase central de la serializacion xml es XmlSerializer y sus métodos son Serialize y Deserialize
- Para serializar se debe tener un constructor por defecto en cada clase a serializar
 - XmlSerializer inicia una nueva instancia , la cual puede serializar objetos del tipo especificado XmlSerilizer(type)
 - Serialize(stream,object) Serializa el objeto y escribe en un xml utilizando el stream indicado.
 - Deserialize(stream) Deserializa el documento Xml por el stream indicado
- XmlTextWriter(path,encoding) Se crea una instancia :
 - Using(XmlTextWriter writer = new XmlWriter("hola.xml",System.Text.Encoding.UTF8))
 - XmlSerializer ser = new XmlSerializer(type())
 - Ser.Serialize(write,dato)
- XmlTextReader(path)
 - Using(XmlTextReader reader = new XmlTextReader("hola.xml"))
 - XmlSerializer ser = new XmlSerializer(type())
 - Aux = (Dato)ser.Deserialize(reader)

- Serializacion Binaria
 - o Puede serializar atributos privados y públicos
 - o Las clases deben tener constructor por defecto
 - o Metodos importantes: Serialize, deserialize
 - o BinaryFormatter() Inicializa una nueva instancia.
 - o Serialize(stream,object)
 - o Deserialize(stream)
 - o FileStream Genera un objeto para leer, escribir, abrir y cerrar archivos.
 - o FileStream(path,modo de apertura)
 - o Read(byte[] array, int offset,int count) Lee el bloque de bytes y escribe los datos en el buffer dado
 - o Seek(long offset,origin) Establece la posición en el stream
 - o Write(byte array[],int offset,int count) Escribe un bloque de bytes en el stream

Clases Selladas:

No se puede heredar de una clase sellada

Interfaces

- Es una clase abstracta sin atributos donde todos sus métodos o propiedades son abstractos.
 - o Todos los métodos son públicos(no se permite especificarlo)
 - o Todos los métodos son abstractos, no se permite especificarlos
 - o Se pueden especificar propiedades.
 - o Las clases pueden implementar varias interfaces.
 - o La utilidad de interfaces es la de “simular” la herencia multiple.
- ```
[Modif] interface INombreInterface
{
}
```
- o Para implementar una interface en una clase se utilizan los :
    - [modif] class NombreClase : INombreInterface
  - o Para implementar una interface en una clase derivada, primero se indica la clase base y luego la interface separada por ,
  - o Para sobrescribir miembros de la interface no se utiliza override.

THREADS

Se debe crear una instancia de Thread y pasarle al constructor un método que no retorne nada ni reciba parámetros.

Metodos :

- Start()
- Abort()
- Suspend()
- Resume()      Pone en ejecución nuevamente el thread

## EXTENSION

Nos simplifican la sintaxis de llamada. Representan métodos estáticos como métodos de instancia. Un método de extensión utiliza la palabra clave **this** en su lista de parámetros. Debe estar ubicado en una clase estática. Se debe crear una clase estática con un método estático que reciba como parámetro ( `this TIPO_DE_DATO obj`)

## Clases Genericas

Las clases genéricas encapsulan operaciones que no son específicas de un tipo de datos concreto. Las clases genéricas se utilizan frecuentemente con colecciones como listas vinculadas, tablas hash, pilas, colas, árboles, etc. Las operaciones de agregar y quitar elementos de la colección se realizan básicamente de la misma forma, independientemente del tipo de datos que se van a almacenar.

Cuando se hereda de una clase genérica, se debe especificar el tipo con el que se trabaja:

Derivada<tipo> : Generica<tipo>

## Abstracta

El modificador `abstract` se utiliza para indicar que una clase está incompleta y que sólo se va a utilizar como una clase base. No se pueden instanciar. Los métodos o propiedades declarados como abstractos no se deben implementar, estos son implementados como override en la clase derivada.

## Polimorfismo

El *polimorfismo* se refiere a la posibilidad de definir múltiples clases con funcionalidad diferente, pero con métodos o propiedades denominados de forma idéntica, que pueden utilizarse de manera intercambiable mediante código cliente en tiempo de ejecución. Se declaran métodos como virtuales en la clase base, si se quiere dar una funcionalidad diferente, estos deben ser declarados como override en las clases derivadas.