Programacion II recuperatorio primer parcial

# Conversiones basicas

* Implicitas: No interviene el programador.
* Explicitas: Interviene porque puede haber perdida de datos.

# Clase

Una clase es una Clasificación. Es una abstracción de un objeto. Incluye datos y métodos (definen comportamiento). Es una estructura de datos.

* Es una construcción **Estatica** que describe:
  + Comportamiento común.
  + Atributos(estado).

# Modificadores

* Son los que determinan la visibilidad de la clase/método/etc. Por defecto son private.

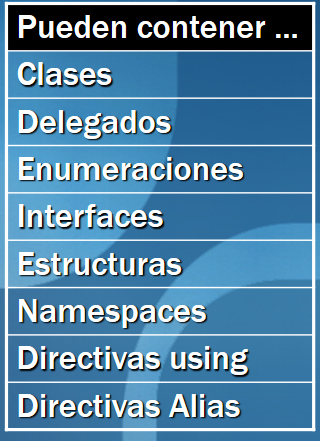
# NameSpace

* Es una agrupación lógica de clases y otros elementos. Toda clase está dentro de un NameSpace
* Proporcionan un marco de trabajo jerárquico sobre el cual se construye y organiza todo el código.
* Su función principal es: la de la organización del código para reducir los conflictos entre nombres.

## Directivas de un NameSpace

* Son elementos que permiten a un programa identificar los NameSpace que se usaran en el mismo. C# posee dos directivas **Using** (permite la especificación de una llamada a un método sin el uso obligatorio de un nombre completamente cualificado) y **Alias** (permite darle un nombre distinto a un namespace).

## Miembros de un NameSpace



# OBJETOS y Variables

Objetos:

* Son instancias de una clase.
* Son creados en tiempo de ejecución.
* Poseen Comportamiento (**Métodos: tareas específicas**) y Estado (**Atributos: Permite almacenar información fija o variable**). Se utiliza punto (.) para acceder a estos.
* Se utiliza la palabra reservada **NEW** para crear objetos.

**LAS VARIABLES TIENEN DESTRUCCION DETERMINISTA (se sabe el inicio y final)**

Se crea en el momento de declararla y se destruye al final del ámbito en el que está declarada

El tiempo de vida de una **variable local** está vinculado al ámbito en el que se declara. En general tiene **tiempo de vida corto** (muere al finalizar el método)

**LOS OBJETOS TIENEN DESTRUCCION NO DETERMINISTA (pueden seguir siendo usados fuera del método)**

El tiempo de vida de un **objeto** no está vinculado a su ámbito. **Tiempo de vida más largo. Destrucción no determinista.**

El Stack es liberado automáticamente y el Heap es adminsitrado por el Garbage Collector.

**NO ES POSIBLE DESTRUIR OBJETOS DE FORMA EXPLICITA, SE DESTRUYEN POR RECOLECCION DE BASURA (Garbage Collector). ESTE BUSCA OBJETOS INALCANZABLES Y LOS DESTRUYE. LOS CONVIERTE DE NUEVO EN MEMORIA. NORMALMENTE LO REALIZA CUANDO EMPIEZA A FALTAR MEMORIA.**

## Ciclo de vida de un objeto

* Se utiliza un constructor para inicializar un objeto en memoria.
* Se utiliza sus métodos y/o atributos.
* Se vuelve a convertir el objeto en memoria, liberando esta.

## CLR (Common Language Runtime)

* Define un conjunto común de tipos de datos orientados a objetos.
* Todo tipo hereda directa o indirectamente del tipo System.Object

El CLR adminsitra dos segmentos de memoria: Stack o Pila y Heap o Montón.

* Stack o Pila: Se almacenan los **tipos VALOR**. Las variables se asignan en los Stack, esto quiere decir que no puede utilizarse fuera del ámbito en donde se crea. Se comportan como una lista LIFO (last In – First Out) donde se van apilando valores uno encima del otro y solo se puede recuperar un valor desapilando los que tiene por encima. Tanto su referencia como su valor se encuentran en la misma posición de memoria. (EJ: caracteres, números enteros,etc). La memoria ocupada es liberada automáticamente por el CLR una vez que se finaliza el procedimiento o bloque de código. Es por esto el ciclo de vida corto de una variable.
* Heap(monton): Se almacenan **los tipos REFERENCIA**. Se llaman asi porque su almacenamiento se encuentra dividido 🡪En el stack se alamacena una referencia al contenido de la variable.

🡪En el heap se guarda el valor de la variable.

Los objetos se almacenan en la memoria del Heap.

## Constructores

* Los constructores **son métodos especiales** que se utilizan para inicializar objetos al momento de su creación.
* La única forma de crear un objeto es mediante la palabra reservada new para adquirir y asignar memoria.
* Siempre llevan el mismo nombre de la clase.
* Siempre existe un constructor por defecto (es el que inicializa todos los campos a cero, null, false)
* Tipos de constructores:
* Constructores de instancia: que inicializan los objetos.
* Constructores estáticos: que son los que inicializan las clases.
  + - * Solo inicializará los atributos estáticos.
      * No debe llevar modificadores de acceso.
      * Usan la palabra **STATIC**.
      * No pueden recibir parámetros.

# Llaves {}

Son marcadores que además de indicar el número de parámetro que se usará podemos indicar la forma en que se mostrará.

Cuantos caracteres se mostrarán y si se formatearán a la derecha o izquierda o también pueden indicar otros valores de formato.

Se utiliza de la siguiente forma:

**{ N [, M ] [:Formato] }**

Donde:

**N** será el número del parámetro, empezando por cero.

**M** será el ancho usado para mostrar el pará metro, el cual se rellenará con espacios. Si M es negativo, se justificará a la izquierda y si es positivo se justificará al a derecha.

**Formato** será una cadena que indicará un formato extra a usar con ese parámetro.

# Sobrecargas

Sobrecarga de Métodos:

* Los métodos se sobrecargan cambiando el número y el tipo de parámetros (cambiando la firma del método). Son difíciles de depurar y de mantener.
* El compilador distingue entre métodos sobrecargados comparando la lista de parámetros.
* La función de la sobrecarga consta en que si hay métodos similares que reciben distintos parámetros o si se quiere añadir funcionalidad a un código existente.

Sobrecarga de método constructor: (respetando los mismos principios para los métodos).

• **Instancia**:

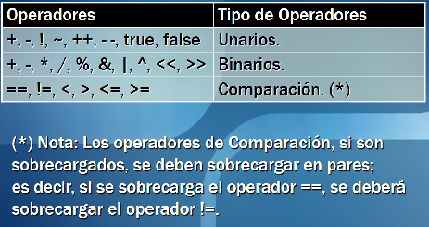
Útil para determinar diferentes maneras de inicializar los atributos de instancia de una clase.

Solo en estos casos, se puede utilizar el this, para ahorrar código, y dirigir la ejecución del método hacia aquél que recibe primero los parámetros definidos con anterioridad (indicados en el this).

• **Estático**:

Se ejecuta cuando se crea la 1era instancia, por única vez (por ello se ejecuta a lo sumo). Inicializa los valores fijados para los atributos estáticos de la clase.

Sobrecarga de Operadores:



* Existe dos tipos de operadores de conversión:
* Implicitos: Declaración: public static implicit operator tipo (tipo a). Son más fáciles de usar. No son vistos en main.
* Explicitos: Declaración: public static explicit operator tipo(tipo a). No se utilizan constructores, ni atributos. El código es más fácil de entender.

# Pilares de la Programación Orientada a Objetos

* Herencia
* Polimorfismo
* Encapsulamiento
  + Es la que denota la capacidad del objeto de responder a peticiones a través de sus **métodos o propiedades** sin la necesidad de exponer los medios utilizados para llegar a brindar estos resultados.
* Abstracción:
  + Ignorancia selectiva
  + Decide que es importante y que no lo es
  + Se enfoca en lo que es importante.
  + Ignora lo que no es importante.
  + Utiliza la encapsulación para reforzar la abstracción.

ARRAYS Y COLECCIONES

## Arrays

* Puede ser unidimensional, multidimensional o anidado.
* El valor por defecto de elementos de un array es 0 mientras que de los objetos dentro es null.
* Son reference type.
* Implementan la interfaz IEnumerable por lo tanto se puede iterar usando foreach.
* Los elementos de un Array pueden ser de cualquier tipo.
* Los Arrays de C# son todos dinámicos, y antes de poder usarlos habrá que instanciarlos, como si fuera cualquier otro objeto.
* Declaración Unidireccionales: [acceso] Tipo [] nombre\_Array = new Tipo[CANT\_ELEMENTOS];
  + Metodos:
  + CopyTo: copia un array a otro.
  + GetLength: obtiene el número de elementos.
  + Initialize: Inicializa todos los elementos del Array, llamando al constructor por defecto.
  + Propiedades:
  + Lenght: obtiene la cantidad de elementos.
* Declaración Multidimensionales: [acceso] Tipo [] nombre\_Array = new Tipo [FILAS, COLUMNAS];

## Colecciones

* .NET ofrece muchas clases para almacenamiento de datos mucho más especializadas que los Arrays. Estan definidas en los NameSpace System.Collections o System.Collections.Generic.
  + Pilas (LIFO): STACK. Representa una colección de objetos no genéricos del tipo LIFO (Last In FirstOut).

Metodos:

* + - Push (Inserta un objeto a la cima de la pila).
    - Peek: Retorna el elemento de la cima de la pila, sin removerlo.
    - Pop: Idem anterior, pero remueve el objeto (desapila).
  + Colas (FIFO): QUEUE. Representa una colección de objetos no genéricos del tipo FIFO (First In FirstOut).

Metodos:

* + - Dequeue: Retorna y remueve el primer elemento de la cola.
    - Enqueue: Inserta un objeto al final de la cola.
    - Peek: retorna el primer elemento de la cola, sin removerlo.
  + Listas (ArrayList): La colección ArrayList puede cambiar su tamaño dinámicamente según sea requerido.

Metodos:

* + - * Add (Agrega un objeto al final de la colección).
        + Remove (Saca el objeto).
        + Reverse (invirte el orden).
  + Hash Tables: 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 específica.
    - Clear: Remueve todos los elementos de la colección.
    - Remove: Remueve un objeto con una clave específica.

## Genericos

* Usado muy comúnmente para crear clases de colecciones.
* Beneficios:
* Aumenta la seguridad de tipos de datos (errores en tiempo de compilación en lugar de ejecución).
* Aumenta la reutilización de código (habría que crear una clase por cada tipo para obtener los mismos beneficios de performance).
* Aumenta la performance (boxing/unboxing)

**COLECCIONES GENERICAS:**

* Dictionary<Tclave, Tvalor>: Representa una colección de clave-valor.
* List<T>
  + Representa una lista de objetos fuertemente tipada, que puede ser accedida por índice. Provee métodos para buscar, ordenar y manipular listas.
* SortedDictionary<Tclave, Tvalor>
  + Ídem Dictionary, pero son ordenados por la clave.
* SortedList<Tclave, Tvalor>
  + Ídem List, pero son ordenados por la clave.

## Como elegir

* Una lista secuencial donde los valores se descartan una vez leídos...
  + Cola (Queue)
  + Pila (Stack)
* Acceso a sus elementos en un determinado orden...
  + Queue (FIFO)
  + Stack (LIFO)
  + LinkedList (Random, en ambas direcciones)
* Acceso a un elemento por un índice...
  + ArrayList o StringCollection
  + HashTable, SortedList, ListDictionary, StringDictionary (acceso por clave)
  + NameValueCollection, KeyedCollection, SortedList (acceso por clave o índice)
* Un ordenamiento diferente al que fueron ingresados...
  + HashTable (por hash de la clave), SortedList (implementación de IComparer, ordena por clave), ArrayList (IComparer).
* Búsquedas rápidas...
  + ListDictionary es más rápido que HashTable para pequeñas cantidades (10 o menos), SortedDictionary es más rápida que Dictionary (generic) para strings.