Clases y Objetos

Una clase es un **molde** o **plantilla** que almacena las **propiedades** y **comportamientos** de un **conjunto** de **objetos** creados a partir de ella, un objeto en cambio es una **instancia** de esa clase que permite **utilizar** los comportamientos (Métodos) y propiedades (atributos) de la clase

Funciones:
Clases: Agrupan **datos** y **comportamientos** de una sola **entidad**, permiten reutilizar código de forma **ordenada** y son la **base** para aplicar los principios de la POO

Objetos: Representan **entidades** reales y permiten **trabajar** con varias **instancias** independientes pero que tienen la **misma estructura**

Sus ventajas son que permiten utilizar los principios de la POO y otras cosas como reutilización de código, organización, encapsulación, etc…

Diagrama de clases

Un **diagrama de clases** es una **representación gráfica (UML)** del diseño de un sistema orientado a objetos. Muestra las **clases**, sus **atributos**, **métodos** y las **relaciones** entre ellas. Se utiliza para **modelar el sistema antes de programarlo** y para **visualizar cómo interactúan sus componentes**.

Referencia This

Referencia al objeto actual, es decir, al objeto que está ejecutando el código en ese momento, guarda en todo momento la referencia de un objeto de la clase, de esta forma es posible identificar el conjunto de propiedades asociadas a cada objeto.

Constructor y Constructor SobreCargado

Un **constructor** es un método especial de una clase que se ejecuta automáticamente al crear un objeto y se utiliza para **inicializar sus atributos**.
Un **constructor sobrecargado** es cuando se **declaran múltiples constructores** dentro de la misma clase, pero con **diferentes tipos o cantidades de parámetros**, lo que permite **crear objetos de distintas formas según lo que se necesite**.

Garbage Collector

Es un **mecanismo automático** que se encarga de **liberar la memoria** ocupada por **objetos** que ya no se usan (Que ya no tienen más **referencias activas**), **evitando** que el programador tenga que librar **memoria manualmente**

Funcionamiento:

1 - Cuando se **crea** un objeto, se **guarda** en memoria dinámica
2 - Si ya no se **puede acceder** a ese **objeto** porque **ninguna variable lo referencia**, el garbage collector lo marca como basura
3 - El recolector se activa automáticamente y libera su memoria

Para que este funcionamiento se cumpla debemos utilizar la clase GC en .NET la cual permite interactuar con el garbage collector y sus métodos:

GC.Collect(): Fuerza al garbage collector a **ejecutar la recolección de basura**
GC.WaitForPendingFinalizers(): Espera a que se terminen de ejecutar todos los finalizadores pendientes.
GC.SuppressFinalize(obj): Evita que se llame al destructor de un objeto.

Constructores y Destructores

Un **constructor** es un método especial de una clase que se ejecuta **automáticamente al crear un objeto**, y se utiliza para **inicializar sus atributos** o realizar acciones necesarias en el momento de la creación.
Un **destructor** es un método especial que se ejecuta **automáticamente cuando un objeto es destruido** por el Garbage Collector, y se utiliza para **liberar recursos**, como archivos abiertos o conexiones.

Tienen el mismo nombre que la clase pero precedido del símbolo ~
No tienen argumentos
No tienen modificadores de alcance

Atributos Estáticos

Un **atributo estático** es una variable que pertenece a la **clase** en lugar de a los **objetos**. Se **comparte** entre todas las **instancias** y se **accede** mediante el **nombre** de la clase. Se usa para **almacenar** información **común** o **global** que afecta a todos los objetos de esa clase. Los métodos estáticos sólo pueden hacer uso de otros métodos y atributos estáticos.

Ventanas de Diálogo

Tienen la característica de **detener** la **ejecución** del código de la aplicación hasta que se **cierran**. Cuando se cierran **devuelven** un valor que indica la **forma** en la que se **cerró**. Los valores **posibles** están definidos como **constantes** de la clase DialogResult(), algunos de los valores posibles son: OK, Cancel, Abort, Ignore, No, None, Retry y Yes

Ya que las ventanas de diálogo son clases, es necesario crear un objeto de esa clase, hacerlo visible y finalmente destruirlo manualmente con los métodos ShowDialog() y Dispose().

Conceptos de las POO

Asociación simple es una relación **estática** entre dos clases que indica que una **clase usa o conoce a otra**, no implica propiedad ni control fuerte, Este vínculo se implementa en una de las clases declarando un atributo que hace referencia a un elemento de la otra clase. La relación queda establecida en el diagrama UML por una línea de asociación. Y se especifica su definición sobre la línea de la relación (bidireccional o unidireccional)

Agregación es una relación “Tiene un” de tipo débil, representa una relación entre pares, un objeto puede pertenecer a otro, pero puede existir por sí solo, se representa en UML con un rombo vacío

Composición es una relación fuerte de **propiedad** y **ciclo de vida**, expresa una relación de **pertenencia** entre las **partes y el todo**, el ciclo de vida del todo esta directamente **ligado** al ciclo de vida de las partes, también es una relación “Tiene un” y si uno de los objetos se destruye, ambos se destruyen, se representa en UML con un rombo relleno

Características:
Composición

Relación fuerte “TIENE UN”.

- Los ciclos de vida están íntimamente ligados (se crean y destruyen juntos)
- La parte que representa el todo es responsable de crear el objeto parte.
- Las partes se co-crean con el todo.
- Cuerpo humano (todo) tiene : cabeza, tronco, extremidades (partes).
- Una Factura (todo) tiene : ítem (parte).

Agregacion

- Relación débil “TIENE UN”.
- • Los ciclos de vida no están ligados. • Las partes y el todo “NO se co-crean”. Pueden existir los objetos por separado o no. Se pueden crear los objetos en momentos distintos.
- La parte que representa el todo, es responsable de relacionar el objeto parte que ya debe existir previamente. La responsabilidad de crearlo no es del todo.
- Sistema Comercial (todo) tiene: clientes, productos, facturas (partes) y se pueden agregar o eliminar en el ciclo de vida del comercio.

Colección ArrayList

Es una clase que **define objetos** en una **estructura dinámica**, en forma de **arreglo**, pero de tamaño **dinámico**, por lo que pueden **ampliarse y reducirse** en tamaño, **no es posible asignar** un tipo específico a estos arreglos, dado que **almacenan referencias** a objetos, es posible alojar **cualquier tipo de objeto en la estructura**, pertenece a System.Collections

Metodos de Interés:
Add: **Agrega** un nuevo objeto a la colección y **retorna** un valor entero que identifica el objeto en conjunto

Capacity: Retorna la cantidad de objetos que se pueden almacenar en la colección sin reasignar memoria

Clear: Elimina todos los objetos en la colección

Contains: Toma como argumento la referencia a un objeto y retorna el valor true si se tiene almacenada esa referencia en la colección o false en caso contrario

Count: Retorna la cantidad de referencias almacenadas

IndexOf: **Busca** una referencia en la colección en la posición indicada por uno de sus argumentos y retorna el índice de su posición

Insert(): Inserta una referencia en la posición indicada dada por argumento

Remove: Eliminar a primera ocurrencia de una referencia en la colección

RemoveAt: Elimina un objeto identificando por su posición, la cual debe ser dada como argumento.

TrimToSize: Reduce el tamaño de la colección al valor mínimo de elementos que contenga

Herencia/Especialización

Es un **mecanismo de la Programación Orientada a Objetos (POO)** que permite crear nuevas clases llamadas **subclases o clases derivadas** a partir de una clase existente, llamada **clase base o padre**, **heredando sus atributos y métodos**.
Esto permite **reutilizar código**, **extender funcionalidades** y aplicar **polimorfismo**.
La relación que se establece es del tipo **"es un"** (por ejemplo, *un perro es un animal*).

Especialización

Es un **proceso de diseño** donde se parte de una **clase más general** y se crean clases más específicas que **agregan o modifican comportamientos** para representar casos particulares.
La especialización se logra mediante **herencia**, y permite adaptar una estructura común (la clase base) a distintas variantes o subtipos.
Es el proceso **inverso a la generalización**: en lugar de abstraer lo común, se **detallan las diferencias**.

Generalización

Procesos mediante el cual se **identifica** lo común entre varias clases y se extrae en una clase base o superclase, para evitar duplicación y facilitar la reutilización, evitando la redundancia… es el proceso inverso a la especialización: mientras la especialización crea subclases más específicas, la generalización abstrae características comunes hacia una clase más general.

Interfaces

Permiten lograr **herencia de comportamiento**, esto es lograr que un conjunto de objetos diferentes clases **respondan** a un conjunto de métodos sin necesariamente ser todos del mismo tipo común

Herencia y Constructores

Cuando una clase derivada se crea, también se ejecuta el constructor de su clase base, esto puede pasar de dos formas

Explícita: La clase **derivada** llama directamente al constructor de la base usando la palabra base y pasa los parámetros necesarios

Implicita: Si no se llama explícitamente, se ejecuta automáticamente el constructor sin parámetros de la clase base.

Si la clase base también hereda de otra, se forma una cadena de constructores que se ejecutan desde la clase base más general hasta la más específica, esta cadena termina en la clase object que es la raíz de todas las clases en C#, osea, los constructores se ejecutan en orden desde la clase base hasta la clase derivada, y el constructor de la derivada es el último en ejecutarse

Herencia y clase object

Todas las **clases heredan** de la clase llamada **object**, esta **define** varios métodos básicos que todas las clases heredan y pueden redefinir si quieren, uno de los métodos más importantes es ToString() que devuelve una representación en texto del objeto, podemos sobreescribir ToString() en nuestras propias clases para mostrar información útil del objeto, otros métodos importantes que vienen de object son: Equals, Finalize, GetHashCode, GetType,MemberWishCode, ReferenceEquals.

Herencia y Sealed

En herencia se pueden redefinir métodos de la clase base en las clases derivadas usando virtual y override, desde la implementación nueva de un método, se puede llamar a la versión original en la clase base usando la palabra clave base, el modificador sealed se usa para impedir que un método o clase se sobreescrito o heredado más allá de ese punto, cuando un método está marcado como sealed, no puede ser redefinido en clases que hereden de esa clase derivada, por defecto, los métodos estáticos y los métodos privados ya tienen esta propiedad: no pueden ser sobreescritos

Redefinición de Métodos

Cuando una clase hija sobreescribe un método de su clase padre usando override, se está redefiniendo ese método, aunque se use una variable del tipo de la clase base, si el objeto real es de la clase derivada, se ejecuta el método de la clase derivada, esto se llama polimorfismo: el comportamiento se adapta según el tipo real del objeto, no según el tipo de la variable

Clases Abstractas y Concretas

Una clase abstracta es una clase que no se puede usar para crear objetos directamente. sirve como modelo base para que otras clases hereden de ella, no se puede instanciar, puede tener métodos abstractos que las clases hijas deben implementar y también pueden

tener métodos normales, también pueden tener propiedades abstractas, sirven para obligar que las clases hijas implementan ciertos comportamientos y aplicar polimorfismo

Interfaces del Programador

Una interfaz declarada por el programador solo incorpora la declaración de métodos con su firma, no su implementación y no puede incluir estado.

Tratamiento de Textos

Posee herramientas para tratar porciones de texto como:

String: Pertenece a System, permite la representación estática de cadenas de texto literales,provee métodos de comparación y localización de texto, agregar, quitar o reemplazar caracteres entre otros, es inmutable (No se puede modificar su contenido una vez creado), si se modifica un string, se crea una nueva cadena en memoria.

Métodos String:

IndexOf	Busca la primera aparicion de un caracter o subcadena y devuelve su indice
LastIndexOf	Busca la ultima aparicion de un caracter o subcadena y devuelve su indice
IndexOfAny	Busca la primera aparicion de un conjunto de caracteres en la cadena y devuelve su indice
LastIndexOfAny	Busca la ultima aparicion de un conjunto de caracteres en la cadena y devuelve su índice
StartsWith	Determina si la cadena comienza con una subcadena especificada
EndsWith	Determina si la cadena termina con una subcadena especificada
PadLeft	Rellena la cadena a la izquierda con espacios en blanco o un caracter especificado
PadRight	Rellena la cadena a la derecha con espacios en blanco o un caracter especificado

SubString: Se utiliza para extraer una parte de una cadena, devuelve una nueva cadena que comienza en una posición específica de la cadena original.

SubString - Argumentos:
int startIndex: La posición inicial desde donde se comenzará a extraer la subcadena
int length: La longitud de la subcadena que se desea extraer, no es obligatorio ya que si no se usa, se extrae la subcadena desde el método startIndex hasta el final de la cadena

Trim: Es utilizado para eliminar los espacios en blanco al principio y final de la cadena, devuelve la cadena con los caracteres especificados eliminados de ambos extremos.

Trim - char[] trimChars: Especifica un arreglo de caracteres que se desea eliminar, si no se especifica, el método elimina los espacios en blanco

Interfaces String:

IEnumerable - GetEnumerator()	Devuelve un enumerador de la colección de caracteres en una cadena
IComparable - CompareTo(Object obj)	Compara el objeto actual con otro objeto y devuelve el valor que indica si el objeto es anterior, igual o posterior a la otra
IEquatable - Equals(String other)	Determina si la cadena actual es igual a otra cadena
IFormattable - ToString()	Devuelve una representación de cadena de la instancia actual

StringBuilder: Pertenece a System. Clase que permite la representación dinámica de cadenas, es decir que una cadena puede modificarse sin crear un nuevo objeto, provee métodos para agregar, insertar, reemplazar y remover caracteres, además permite especificar ciertos formatos personalizados, es mutable por lo que puede modificar textos muchas veces sin crear nuevas cadenas

Métodos StringBuilder:

`Append(string texto)`	Agrega texto al final del contenido actual.
`AppendLine(string texto)`	Agrega texto y luego un salto de línea.
`Insert(int índice, string texto)`	Inserta texto en una posición específica.
`Remove(int índice, int longitud)`	Elimina una cantidad de caracteres desde una posición dada.
`Replace(string viejo, string nuevo)`	Reemplaza todas las ocurrencias de un texto por otro.
`Clear()`	Elimina todo el contenido del `StringBuilder`.
`ToString()`	Convierte el contenido actual a un `string` normal.

Interfaces StringBuilder:

ISerializable - GetObjectData()	Llena una objeto SerializationInfo con los datos necesarios para deserializar el objeto StringBuilder actual

Regex: Pertenece a System.Text.RegularExpressions provee métodos que permiten utilizar expresiones regulares para tratar cadenas permitiendo buscar, validar o reemplazar patrones de texto

Métodos Regex:

Método	Descripción
`IsMatch(string texto)`	Devuelve `true` si el texto coincide con el patrón. Se usa para validación.
`Match(string texto)`	Devuelve el **primer match (coincidencia)** encontrado en el texto.
`Matches(string texto)`	Devuelve **todas las coincidencias** del patrón en el texto (colección de `Match`).
`Replace(string texto, string reemplazo)`	Reemplaza los valores que coinciden con el patrón por otro texto.
`Split(string texto)`	Divide una cadena usando el patrón como separador.

Excepciones

Es una situación no esperada al normal funcionamiento de la aplicación, los programadores debemos generar excepciones cuando detectamos una condición anómala en el entorno que impida que nuestra aplicación continúe su ejecución de modo normal utilizando las cápsulas TRY y CATCH

Try: es un bloque en el que va todo el código del programa al ejecutarse, si alguna de las líneas del código genera una excepción, se detiene y no se procede con el resto del código pasando al bloque Catch

Catch: Se crea una objeto cuya clase coincide con el tipo de excepción lanzada, se ejecuta el fragmento de código vinculado a ese bloque

Finally: Permite establecer un fragmento de código que se ejecutará siempre, más allá que se lance o no una excepción, es ideal para liberación de recursos

Excepciones Propiedades

Son clases de objetos, al dispararse una excepción se crea un objeto de la clase adecuada, existen dos clases derivadas de la clase Exception de las cuales derivan las demás

SystemException: Excepciones graves de las que no se puede volver, ej: tratar de invocar un método de una variable de referencia que no referencia un objeto o cuando se invoca un método recursivo que llena el stack (StackOverFlowException)

ApplicationException: Representa excepciones que se pueden producir en una aplicación durante su ejecución y sirve como clase base para que los programadores desarrollen clases derivadas para lanzar excepciones.
Message: Propiedad de tipo string que contiene un mensaje asociado a la excepción

StackTrace: Propiedad del tipo string que contiene la lista de todos los métodos invocados y abiertos al momento que la excepción se dispara

InnerException: Es una excepción disparada por otra excepción, si esta propiedad tiene el valor null, indica que se trata de una excepción que no se disparó a raíz de otra

Excepciones Definidas por el Usuario

Al desarrollar aplicaciones, a veces es necesario crear excepciones personalizadas para ofrecer información detallada sobre el motivo de la excepción. Para esto es necesario crear una clase que hereda de ApplicationException, siguiendo la convención de incluir Exception en el nombre y se implementan al menos 3 constructores

Uno predeterminado, inicializa el mensaje asociado a la excepción a través de invocar el constructor de la clase base

Uno que tome un argumento de tipo string, este argumento debe usarse para inicializar también la propiedad message

Uno que tome un argumento de tipo string y un objeto de la clase Exception para lanzar una InnerException

Archivos

Nos permiten persistir la información tratada por nuestras aplicacion
Se identifican a partir de su nombre - extensión y ruta donde se alojan, estas ventanas se incorporan en nuestras aplicaciones a través de los controles OpenFileDialog y SaveFileDialog que se encuentran en el cuadro de herramientas, el nombre del archivo será accesible por la propiedad FileName del control que se trate, esta propiedad de tipo string, contendrá la ruta, nombre y extension indicado por el usuario

Las operaciones con archivos se realizan considerando que son flujos de datos, se utiliza la clase FileStream para conectar el archivo físico con un flujo, después se usan StreamReader o StreamWriter para leer y escribir en el archivo como si fuera un flujo de datos, utilizando métodos como Read, ReadLine, Write y WriteLine.

Propiedades de los Cuadros OFD y SFD

Title: Obtiene o establece el título del cuadro de diálogo de archivos.

CheckFileExists: Obtiene o establece un valor que indica si se muestra una advertencia si el usuario especifica una ruta que no existe.

Filter: Obtiene o establece la extensión de nombre de archivo predeterminado.

FilterIndex: Obtiene o establece el índice de filtro seleccionado actualmente en el cuadro de diálogo de archivos.

InitialDirectory: Obtiene o establece el directorio inicial mostrado por el cuadro de diálogo de archivos

Clase File

Copy: Copia un archivo existente o un nuevo archivo

Exists: Determina si existe el archivo especificado

Delete: Borra el archivo especificado

Move: Mueve un archivo a una nueva ubicación

ReadAllLines: Abre un archivo de texto, lee todas las líneas y lo cierra

WriteAllLines: Crea un archivo, escribe una colección de strings en el archivo y lo cierra

Clase FileStream

Proporciona un Stream (flujo) para un archivo lo que permite operaciones de lectura y escritura

Clase Directory

CreateDirectory(String): Crea todos los directorios y los subdirectorios en la ruta de acceso especificada, a menos que ya existan.

Exists: Determina si la ruta de acceso existe en el disco

Delete(String, Boolean): Elimina el directorio especificado y los subdirectorios que contenga.

Move: Mueve un directorio y su contenido a una nueva ubicación.

FileMode - FileAccess

FileMode

Append: Abre el archivo si existe y realiza una búsqueda hasta el final del mismo, o crea un archivo nuevo, solo se puede utilizar junto con FileAccess.Write. Al intentar realizar una

búsqueda hasta una posición antes del final del archivo se producirá la excepción IOException

Create: Específica que se debe crear un archivo nuevo, si ya existe, se sobrescribe, si el archivo ya existe pero está oculto, se produce una excepción UnauthorizedAccessException.

CreateNew: Específica que debe crear un archivo nuevo, si ya existe, se produce una excepción IOException.

Open: Específica que se debe abrir un archivo existente, si no existe se produce una excepción FileNotFoundException.

OpenOrCreate: Específica que se debe abrir un archivo si ya existe, si no existe, debe crearse uno nuevo. Si se abre el archivo con FileAccess.Read, se requiere el permiso FileIOPermissionAccess.Read. Si el acceso a archivos es FileAccess.Write, se requiere el permiso FileIOPermissionAccess.Write. Si se abre el archivo con FileAccess.ReadWrite, se requieren ambos permisos.

Truncate: Específica que se debe abrir un archivo si ya existe, cuando se abre, debe truncarse para que su tamaño sea de 0 bytes. Requiere el permiso FileIOPermissionAccess.Write. Al intentar leer un archivo abierto con FileMode.Truncate, se produce una excepción ArgumentException

FileAccess

Define constantes de acceso de lectura, de escritura y de lectura/escritura para un archivo.

Read: Acceso de lectura al archivo, se combina con Write para obtener acceso de lectura y escritura

ReadWrite: Acceso de lectura y escritura al archivo. En este archivo se pueden escribir y leer datos.

Write: Acceso de escritura al archivo. En este archivo se pueden escribir datos. Se combina con Read para obtener acceso de lectura y escritura

StreamReader

Es un lector de texto que lee los caracteres de un flujo de bytes en una codificación determinada, admite constructores a partir de un flujo abierto o de una cadena de caracteres que indica el nombre del archivo de texto

StreamReader(Stream): Inicializa una nueva instancia de la clase StreamReader para el flujo especificado.

StreamReader(String): Inicializa una nueva instancia de la clase StreamReader para el nombre de archivo especificado

StreamWriter

Es un sistema de escritura de texto que escribe los caracteres de una secuencia de bytes en una modificación determinada, admite constructores a partir de un flujo abierto o de una cadena de caracteres que indica el nombre del archivo de texto

StreamWriter(Stream): Inicializa una nueva instancia de la clase StreamWriter para la secuencia especificada usando la codificación UTF8 y el tamaño de búfer predeterminado.
StreamWriter(String): Inicializa una nueva instancia de la clase StreamWriter para el archivo especificado usando la codificación y tamaño de búfer especificados

Serialización

Consiste en lograr una representación del estado de un objeto y de las características del mismo como un conjunto de bytes, los cuales pueden manipularse para ser almacenados, su contraparte es la serialización que consiste en convertir un flujo de bytes en un objeto.

Serialización - Almacenamiento de datos

Los tipos de datos basicos asi como los objetos string son serializables, pero otros objetos pueden no serlo, por esto cuando se desea serializar para persistir el estado de un sistema hay que asegurarse que todas las entidades del mismo puedan ser serializables, esto se logra marcandolas con el atributo [Serializable] en la definición de la clase o que implemente la interfaz ISerializable.

Si los objetos son serializables, podemos hacer persistencia del estado de la misma almacenando el resultado de la serialización en archivos, para esto se utiliza la clase FileStream para poder crear un flujo asociado a un archivo y BinaryFormatter para poder almacenar en el archivo los objetos serializados.

Serialización - BinaryFormatter

Es una clase que está definida en el espacio de nombre
System.Runtime.Serialization.Formatters.Binary
De modo similar, el método Deserialize toma como argumento un objeto de la clase FileStream y retorna un objeto cuyo estado estaba almacenado en el archivo original

El método Serialize de la clase BinaryFormatter toma como argumento el objeto de tipo FileStream y el objeto que se desea serializar y almacenar, Serialize y Deserialize trabajan con un único objeto cuyo estado se almacena o recupera de un archivo, por eso tal objeto debe contener o referenciar los demás objetos cuyos estados interesen guardarse.

Estructuras de Datos

Son algoritmos, desarrollados para resolver problemas específicos, que permiten tratar y almacenar información, las listas enlazadas resuelven el problema de almacenar valores cuya cantidad no es conocida y puede variar dinámicamente, son estructuras de datos lineales similares a los vectores, que nos permiten agregar elementos en cualquier posición, existen diferentes tipos de listas enlazadas y diferentes implementaciones.

Estructuras de Datos - Clases AUTO-REFERENCIADAS

Las clases Auto-Referenciadas tienen como particularidad que contienen una variable de instancia referencia del mismo tipo de la clase, esto consiste en el principio de implementación de las listas enlazadas.

Estructuras de Datos - Listas Enlazadas

Su metodología de trabajo es crear nodos que se implementan como clases y contienen el valor a almacenar y una variable de referencia al siguiente valor, haciendo uso de esta manera de una estructura Auto Referenciada, cuando el valor de la variable de referencia es null, indica que no hay valores más allá del nodo actual.

Estructuras de Datos - Implementación de Listas Enla.

Nodo: Representa un valor en la lista y contendrá una variable de referencia al elemento siguiente.

Lista: Que representará la lista enlazada en sí misma, permitiéndonos agregar valores, accederlos, etc.

Estructuras de Datos - Pilas

Es una estructura lineal de datos, se agregan valores en un extremo y al extraerlos, se los extrae de ese mismo extremo, por eso es de tipo LIFO (Last input, first output), se llama a Push para agregar un elemento a la pila y Pop para sacar un elemento de la pila, Stack es un ejemplo de pila, si se trabaja con una pila dinámica se puede implementar con una lista enlazada, pero no necesariamente todas las pilas son dinámicas ni se deben implementar de esta forma.

Pilas - Stack

Su capacidad es el número de elementos que dicha Stack puede contener, a medida que se agregan elementos, la capacidad aumenta automaticamente segun lo que se requiera, stack acepta null como valor válido y permite elementos duplicados

Propiedades y Métodos:
Count: Obtiene el número de elementos incluidos en Stack.

Clear: Quita todos los objetos de la colección Stack. C

lone: Crea una copia superficial de la colección Stack.

Contains: Determina si un elemento se encuentra en Stack.

CopyTo: Copia Stack en una Array unidimensional existente, a partir del índice especificado de la matriz.

Equals(Object): Determina si el objeto especificado es igual al objeto actual. Se hereda de Object).

GetEnumerator: Devuelve una interfaz IEnumerator para la interfaz Stack.

GetHashCode: Sirve como una función hash para un tipo en particular. Se hereda de Object).

GetType: Obtiene el Type de la instancia actual. Se hereda de Object).

Peek: Devuelve el objeto situado al principio de Stack sin eliminarlo.

Pop: Quita y devuelve el objeto situado al principio de Stack.

Push: Inserta un objeto al principio de Stack.

ToArray: Copia Stack en una nueva matriz.

ToString: Retorna una cadena que representa al objeto actual. H de Object).

Finalize: Permite que un objeto intente liberar recursos y realizar otras operaciones de limpieza antes de ser reclamado por la recolección de elementos no utilizados. Se hereda de Object)

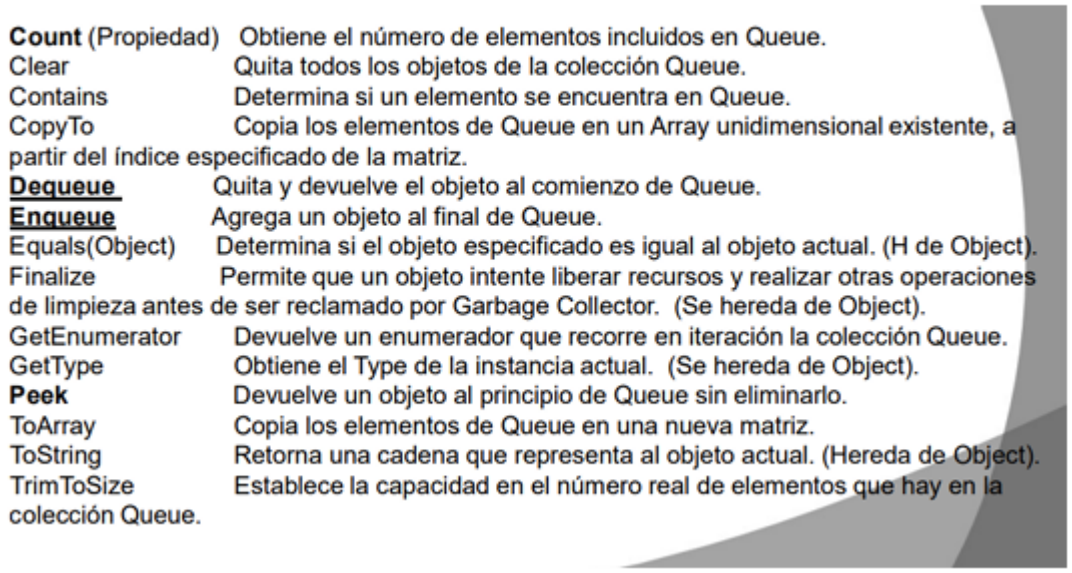
Estructuras de Datos - Colas

Estructura lineal, se agregan valores en un extremo y cuando se extraen, se los extrae del extremo opuesto, por eso se la conoce como FIFO (First Input, First Output), se llama Enqueue a la función a través de la cual se agrega un elemento a la cola y Dequeue a la función que se utiliza para sacar un elemento. Si se trabaja con una cola dinámica se puede implementar con una lista enlazada. Pero no necesariamente todas las colas son dinámicas ni se deben implementar de esta forma.

Colas - Queue

Almacena mensajes en el orden en el que fueron recibidos para el procesamiento secuencial, los objetos almacenados en Queue se insertan en un extremo y se quitan del otro. La capacidad de una colección Queue es el número de elementos que dicha Queue puede contener. A medida que se agregan elementos a Queue, la capacidad aumenta automáticamente según lo requiera la reasignación. La capacidad se puede disminuir si se llama al método TrimToSize. La clase Queue acepta null como valor válido y permite elementos duplicados

▼ COLAS - PROPIEDADES Y METODOS



Estructuras de Datos - System Collections

Este espacio de nombres brinda los métodos que representan el comportamiento de las estructuras de datos vistas.

Listas: Una interface que define métodos a implementar iList(Metodos: Add, Clear, Foreach, RemoveAt, Sort, Reverse)

Colas: La clase Queue permite representar el comportamiento FIFOO, Queue(Metodos: Clear, Enqueue, Dequeue, Peek, Count, First, Last)

Pilas: La clase Stack permite representar el comportamiento LIFO, Stack(Métodos: Clear, Peek, Pop, Push, First, Last)

Estructuras de Datos - Arboles

Estructura no lineal de dos dimensiones, cada nodo contiene referencias a dos o más nodos, si nos referimos a un árbol binario, cada árbol tendrá dos ramas, el nodo raíz es el primer nodo del árbol del cual prenden los demás nodos, cada puntero de cada nodo apunta a dos nodos que constituyen ramas, cada uno de esto almacena la dirección de otras estructuras o NULL

Estructuras de Datos - Arboles Binarios

Se caracterizan por la disposición de elementos que almacenan se distribuyen de manera tal que el elemento ubicado a la izquierda siempre es menor que él ubicado a la derecha, a partir de esto se logran diferentes recorridos sobre los datos:

PreOrden:
Procesar el valor del Nodo
Recorrer el subárbol izquierdo
Recorrer el subárbol derecho

PostOrden:
Recorrer el subárbol izquierdo
Recorrer el subárbol derecho
Procesar el valor del nodo

InOrden:
Recorrer el subárbol izquierdo
Procesar el valor del nodo
Recorrer el subárbol derecho

Colecciones

Son implementaciones de interfaces, algunas son

ICollection: Es la base de la jerarquía de las colecciones

IList: Colección ordenada que puede manipularse como arreglo, incorpora métodos para buscar y modificar una colección.

IDictionary: Permite trabajar con colecciones indexadas por un objeto clave dado. Incorpora un indexador de tipo object que permite identificar los valores

IEnumerable: Incorpora un método GetEnumerator que retorna un objeto IEnumerator que permite acceder a los objetos contenidos en la colección.

Nos interesa analizar las siguientes clases y métodos del espacio de nombres System.Collections

Array Sort Copy BinarySearch)
ArrayList Add Clear Count Capacity Item Insert)
Queue Count Peek Enqueue Dequeue)
SortedList Add Clear Capacity Count Values Item Remove RemoveAt)
Stack Count Peek Pop Push Finalize

Colecciones - Array

Clase abstracta que implementa la interfaz IList, todos los arreglos derivan de esta clase base, tiene métodos estáticos como:

Array.Sort(Arreglo): Ordena los valores almacenados en el arreglo dado como argumentos, debe implementar IComparable

Array.Copy(vectorOrigen, vectorDestino, cantidadValores): Copa el conjunto de valores indicados desde un vector a otro vector

Array.BinarySearch(vectorOrdenado, valorBuscado): Implementa el metodo de busqueda por bisección buscando el valor dado dentro del conjunto de datos contenidos en el arreglo, retorna la posición del valor buscado en caso de encontrarlo, en caso de no encontrarlo retorna un valor negativo, debe implementar IComparable

Colecciones - ArrayList

Implementa la interfaz IList y representa arreglos cuya cantidad de elementos puede crecer o disminuir según se agreguen o quiten elementos, un objeto de esta clase puede almacenar tantos objetos en la colección como en el valor de la propiedad capacity, cuando se han almacenado tantos objetos como el valor de esta propiedad, se alojan los recursos necesarios para duplicar el valor previo, esta clase maneja colecciones de objetos a través de referencias a object, por lo que será necesario realizar conversiones cada vez que se quieran invocar métodos propios del objeto

Métodos:

Add : agrega un nuevo objeto a la colección y retorna un valor entero que identifica el objeto en el conjunto.

Capacity: cantidad de objetos que se pueden almacenar en la colección sin asignar memoria.

Clear: elimina todos los objetos de la colección

Contains: toma como argumento la referencia a un objeto y retorna el valor true si se tiene almacenada esa referencia en la colección o false en caso contrario.

Count: retorna la cantidad de referencias almacenadas.

IndexOf: retorna el índice de la primera referencia al objeto dado como argumento en la colección o 1 en caso contrario.

Insert: inserta una referencia en la colección en la posición indicada por uno de sus argumentos.

Remove: elimina la primera ocurrencia de una referencia en la colección

RemoveAt: elimina un objeto identificándolo por su posición, la cuál debe ser dada como argumento.

Sort: ordena las referencias a los objetos contenidos en la colección. Para esto, los objetos deben implementar la interfaz IComparable

Colecciones - Doblemente Enlazadas, LinkedList

A diferencia de las anteriores, esta clase utiliza genéricos para la implementación de la lista, la lista implementada es doblemente enlazada por lo que es posible recorrer la misma en ambos sentidos, cada nodo de la lista es un objeto de la clase LinkedListNode

Esta clase incorpora 3 atributos
Una referencia al nodo siguiente, al anterior y una al valor almacenado en el nodo, la gestión de los nodos esta dada a través de la clase LinkedList que incorpora los siguiente métodos:

AddFirst y AddLast: Agregan un elemento a la lista al principio y al final respectivamente.

RemoveFirst y RemoveLast: Eliminan el primer y último elemento de la lista respectivamente (no lo retornan)

Para acceder a la información almacenada en la lista es posible, utilizando el método First a partir del primer nodo y luego siguiendo los enlaces del nodo, utilizando foreach, en este caso debe considerarse que la variable utilizada para el recorrido debe ser declarada del mismo tipo que la información almacenada o utilizando un enumerador

Genéricos

Nos permiten implementar clases y métodos que funcionen con cualquier tipo de dato, sin tener que repetir el mismo código para cada tipo, son útiles para reutilizar código sin importar el tipo de datos, para evitar tener que hacer castings innecesarios y para que el código sea más flexible, seguro y limpio, utilizan <T> para representar un tipo que será definido más adelante, cuando se cree el objeto o se llame al método, ejemplo:

```
class Caja<T>  // <T> indica que es un tipo genérico
{
    public T Contenido;

    public void Mostrar() {
        Console.WriteLine("Contenido: " + Contenido);
    }
}
```

Cuando deseamos crear un objeto de una clase como la mostrada, deberemos indicar con qué tipo específico de dato deseamos trabajar

```
Caja<int> miCajaEntero = new Caja<int>();
miCajaEntero.Contenido = 123;

Caja<string> miCajaTexto = new Caja<string>();
miCajaTexto.Contenido = "Hola";
```

Aquí indicamos al compilador que un tipo de dato es un int y otro un string

Las colecciones **genéricas** reemplazan a las **viejas colecciones no genéricas**.
Las nuevas versiones son **más seguras, más eficientes y evitan castings**.
Algunas, como `LinkedList<T>` y `SortedDictionary<TKey, TValue>`, **solo existen en su versión genérica**

Colecciones Genéricas:
List<T>
LinkedList<T>
Queue<T>
Stack<T>
Dictionary<TKey, TValue>: Colección Genérica que almacena pares de clave y valor, cada elemento tiene una clave única y un valor asociado a esa clave.

SortedList<TKey, TValue>: Colección Genérica que almacena pares clave y valor pero las claves están ordenadas y se pueden acceder por índice.

SortedDictionary<TKey, TValue>: Colección Genérica que almacena pares clave y valor pero no se puede acceder por índice, internamente usa una estructura de árbol, también están ordenadas.

HashSet<T>
SortedSet<T>

Colecciones NO Genéricas:
ArrayList
Queue
Stack
HashTable
SortedList