**1.1 – Clasificación de las estructuras de datos.**

**Concepto.**

Una estructura de datos es una clase de datos que se puede caracterizar por su organización y operaciones definidas sobre ella. Algunas veces a estas estructuras se les llama tipos de datos.

Las estructuras de datos son una forma de organizar los datos en la computadora, de tal manera que nos permita realizar unas operaciones con ellas de forma muy eficiente.

Es decir, igual que un array introducimos un dato y eso es prácticamente inmediato, no siempre lo es, según qué estructuras de datos y qué operaciones.

Depende que algoritmo queramos ejecutar, habrá veces que sea mejor utilizar una estructura de datos u otra estructura que nos permita más velocidad.

Por este motivo es interesante conocer algo más que simplemente los arrays o los hashmaps que casi todo el mundo conoce.

**Clasificaciones.**

ESTRUCTURAS LÓGICAS DE DATOS:

En un programa, cada variable pertenece a alguna estructura de datos explícita o implícitamente definida, la cual determina el conjunto de operaciones válidas para ella. Las estructuras de datos que se discuten aquí son estructuras de datos lógicas. Cada estructura de datos lógica puede tener varias representaciones físicas diferentes para sus almacenamientos

ESTRUCTURAS PRIMITIVAS Y SIMPLES:

Son primitivas aquellas que no están compuestas por otras estructuras de datos, por ejemplo, enteros, booleanos y caracteres. Otras estructuras de datos se pueden construir de una o más primitivas. Las estructuras de datos simples que consideramos se construyen a partir de estructuras primitivas y son: cadenas, arreglos y registros. A estas estructuras de datos las respaldan muchos lenguajes de programación.

ESTRUCTURAS LINEALES Y NO LINEALES:

Las estructuras de datos simples se pueden combinar de varias maneras para formar estructuras más complejas. Los dos cases principales de estructuras de datos son las lineales y las no lineales, dependiendo de la complejidad de las relaciones lógicas que representan. Las estructuras de datos lineales incluyen pilas, colas y listas ligadas lineales. Las estructuras de datos no lineales incluyen grafos y árboles.

DATOS ESTATICOS  
Su tamaño y forma es constante durante la ejecución de un programa y por tanto se determinan en tiempo de compilación. El ejemplo típico son los arrays. Tienen el problema de que hay que dimensionar la estructura de antemano, lo que puede conllevar desperdicio o falta de memoria.

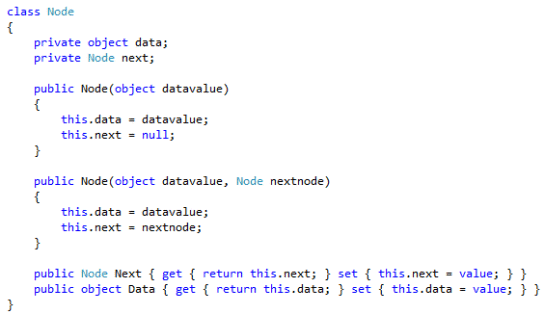
DATOS DINAMICOS

Su tamaño y forma es variable (o puede serlo) a lo largo de un programa, por lo que se crean y destruyen en tiempo de ejecución. Esto permite dimensionar la estructura de datos de una forma precisa: se va asignando memoria en tiempo de ejecución según se va necesitando.

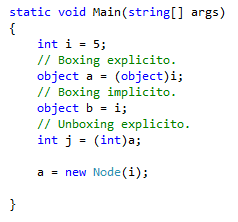
**Ejemplos.**

Clase para definir un nodo

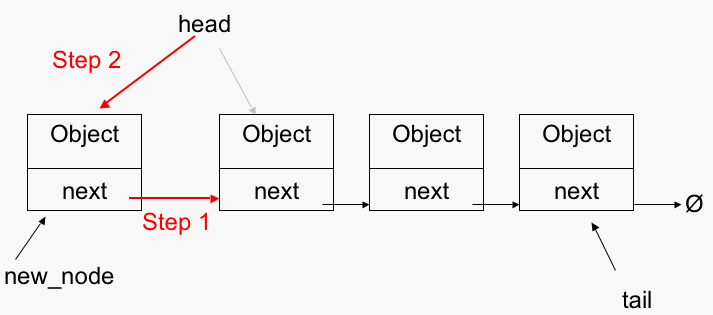
Definimos dos constructores, el segundo permite crear un nodo y definir el nodo al que apunta como siguiente elemento en la lista.



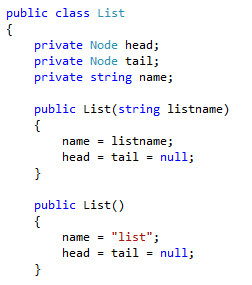
Lo más remarcable de la clase es la referencia object. Nos permite almacenar tipos de datos simples como si fueran objetos. El tipo [object](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/object)es un alias para [Object](https://docs.microsoft.com/en-us/dotnet/api/system.object) en .NET. Todos los tipos de variables son herencia directa de este tipo. Podemos asignar valores de cualquier tipo a variables de tipo object. Cuando una variable se convierte a un tipo object a esta operación se le llama boxing o boxed. Cuando se realiza la operación inversa se denomina unboxing.



Clase lista



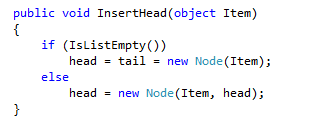
La clase lista tiene contiene como miembros head y tail. Son respectivamente referencias al primer y último nodo de la lista, también definimos una variable string para asignar un nombre a la lista.



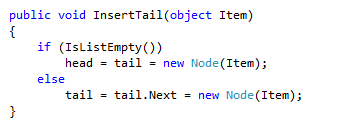
Definimos un método que nos resultará de utilidad más adelante, el método IsListEmpty retorna true si la cabeza de lista apunta a null.



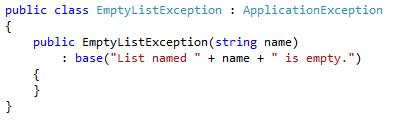
Ahora definimos un método para operar sobre la lista insertando un nuevo nodo al inicio de la misma.  Si la lista esta recién creada o vacía la cabeza y la cola apunta al nuevo y único la misma. En caso contrario la cabeza apunta al nuevo nodo y le pasamos el nodo de cabeza actual como miembro para que quede en segundo lugar.



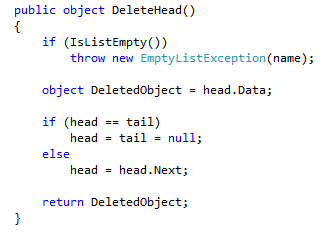
Para añadir un nodo al final definimos el siguiente método:



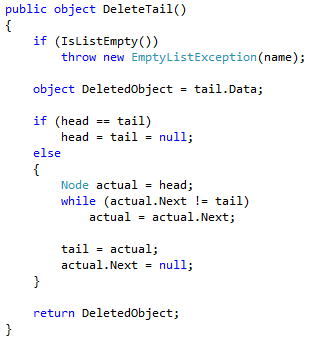
Antes de continuar con un método para borrar un elemento del inicio de la lista vamos a definir una clase EmptyListException para lanzar una excepción cuando se producen operaciones ilegales sobra la lista, por ejemplo, si la lista está vacía. Usamos System.ApplicationException para excepciones generadas por nuestro programa.



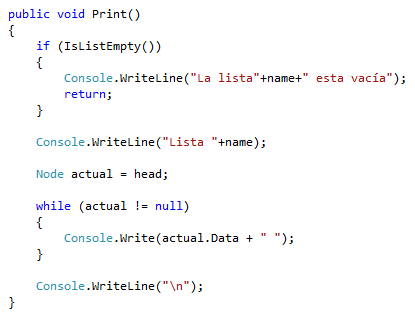
Ahora ya podemos crear un método para borrar un elemento de la cabecera de la lista. Si la lista está vacía lanza una excepción que podemos capturar y tratar desde el programa principal. Después obtenemos el miembro del nodo de cabecera y restablece las referencias del primer y último nodo (si solo hay un nodo en la lista head y last quedaran a null, si hay más de un elemento avanzamos al siguiente nodo la cabecera).



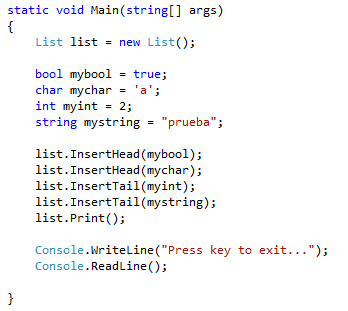
Visto el anterior ejemplo borrar el último nodo es similar. Pero en este caso el método que debemos seguir nodo no es muy eficiente (esto se solucionaría con una lista doblemente enlazada). Recorremos desde el primero nodo uno de detrás de otro hasta que el nodo siguiente no sea el último, de esta manera hacemos que apunte a null quedando fuera el último nodo.



Ahora solo nos queda un método para imprimir los nodos de la lista.



Ahora veamos cómo se puede utilizar:



<http://tesciedd.blogspot.com/2011/01/unidad-1-clasificacion-de-estructuras.html>

<http://decsai.ugr.es/~jfv/ed1/c/cdrom/cap7/cap71.htm>

<https://openwebinars.net/blog/que-son-las-estructuras-de-datos-y-por-que-son-tan-utiles/>