**Tipo de datos abstractos (TDA).**

Un Tipo de dato abstracto (en adelante TDA) es un conjunto de datos u objetos al cual se le asocian operaciones. El TDA provee de una interfaz con la cual es posible realizar las operaciones permitidas, abstrayéndose de la manera en cómo estén implementadas dichas operaciones. Esto quiere decir que un mismo TDA puede ser implementado utilizando distintas estructuras de datos y proveer la misma funcionalidad.

El paradigma de orientación a objetos permite el encapsulamiento de los datos y las operaciones mediante la definición de clases e interfaces, lo cual permite ocultar la manera en cómo ha sido implementado el TDA y solo permite el acceso a los datos a través de las operaciones provistas por la interfaz.

**TDA lista**

Una lista se define como una serie de N elementos E1, E2, ..., EN, ordenados de manera consecutiva, es decir, el elemento Ek (que se denomina elemento k-ésimo) es previo al elemento Ek+1. Si la lista contiene 0 elementos se denomina como lista vacía.

Las operaciones que se pueden realizar en la lista son: insertar un elemento en la posición k, borrar el k-ésimo elemento, buscar un elemento dentro de la lista y preguntar si la lista está vacía.

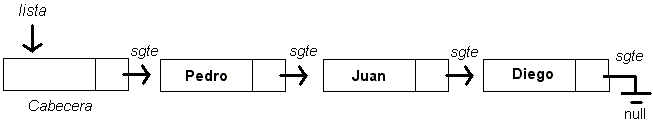
Una manera simple de implementar una lista es utilizando un arreglo. Sin embargo, las operaciones de inserción y borrado de elementos en arreglos son ineficientes, puesto que para insertar un elemento en la parte media del arreglo es necesario mover todos los elementos que se encuentren delante de él, para hacer espacio, y al borrar un elemento es necesario mover todos los elementos para ocupar el espacio desocupado. Una implementación más eficiente del TDA se logra utilizando listas enlazadas.

A continuación, se presenta una implementación en Java del TDA utilizando listas enlazadas y sus operaciones asociadas:

* estaVacia(): devuelve verdadero si la lista está vacía, falso en caso contrario.
* insertar (x, k): inserta el elemento x en la k-ésima posición de la lista.
* buscar(x): devuelve la posición en la lista del elemento x.
* buscarK(k): devuelve el k-ésimo elemento de la lista.
* eliminar(x): elimina de la lista el elemento x.

En la implementación con listas enlazadas es necesario tener en cuenta algunos detalles importantes: si solamente se dispone de la referencia al primer elemento, el añadir o remover en la primera posición es un caso especial, puesto que la referencia a la lista enlazada debe modificarse según la operación realizada. Además, para eliminar un elemento en particular es necesario conocer el elemento que lo antecede, y en este caso, ¿qué pasa con el primer elemento, que no tiene un predecesor?

Para solucionar estos inconvenientes se utiliza la implementación de lista enlazada con nodo cabecera. Con esto, todos los elementos de la lista tendrán un elemento previo, puesto que el previo del primer elemento es la cabecera. Una lista vacía corresponde, en este caso, a una cabecera cuya referencia siguiente es null.



Los archivos NodoLista.java, IteradorLista.java y Lista.java contienen una implementación completa del TDA lista. La clase NodoLista implementa los nodos de la lista enlazada, la clase Lista implementa las operaciones de la lista propiamente tal, y la clase IteradorLista implementa objetos que

permiten recorrer la lista y posee la siguiente interfaz:

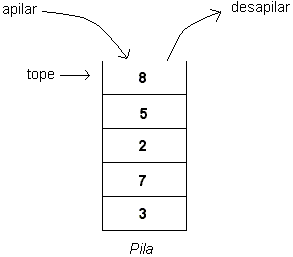
* avanzar(): avanza el iterador al siguiente nodo de la lista.
* obtener(): retorna el elemento del nodo en donde se encuentra el iterador.

Costo de las operaciones en tiempo:

* Insertar/eliminar elemento en k-ésima posición: O(k) (¿Se puede hacer en O(1)?).
* Buscar elemento x: O(N) (promedio).

**TDA pila**

Una pila (stack o pushdown en inglés) es una lista de elementos de la cual sólo se puede extraer el último elemento insertado. La posición en donde se encuentra dicho elemento se denomina tope de la pila. También se conoce a las pilas como listas LIFO (LAST IN - FIRST OUT: el último que entra es el primero que sale).



a interfaz de este TDA provee las siguientes operaciones:

* apilar(x): inserta el elemento x en el tope de la pila (push en inglés).
* desapilar(): retorna el elemento que se encuentre en el tope de la pila y lo elimina de ésta (pop en inglés).
* tope (): retorna el elemento que se encuentre en el tope de la pila, pero sin eliminarlo de ésta (top en inglés).
* estaVacia(): retorna verdadero si la pila no contiene elementos, falso en caso contrario (isEmpty en inglés).

Nota: algunos autores definen desapilar como sacar el elemento del tope de la pila sin retornarlo.

**TDA cola**

Una cola (queue en inglés) es una lista de elementos en donde siempre se insertan nuevos elementos al final de la lista y se extraen elementos desde el inicio de la lista. También se conoce a las colas como listas FIFO (FIRST IN - FIRST OUT: el primero que entra es el primero que sale).



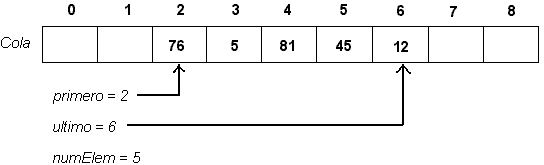
Las operaciones básicas en una cola son:

* encolar(x): inserta el elemento x al final de la cola (enqueue en inglés).
* sacar(): retorna el elemento que se ubica al inicio de la cola (dequeue en inglés).
* estaVacia(): retorna verdadero si la cola está vacía, falso en caso contrario.

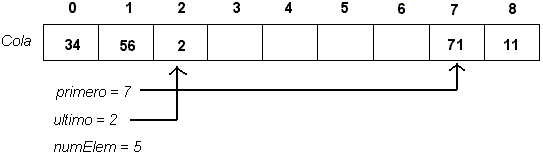
Al igual que con el TDA pila, una cola se puede implementar tanto con arreglos como con listas enlazadas. A continuación, se verá la implementación usando un arreglo.

Las variables de instancia necesarias en la implementación son:

* primero: indica el índice de la posición del primer elemento de la cola, es decir, la posición el elemento a retornar cuando se invoque sacar.
* ultimo: indica el índice de la posición de último elemento de la cola. Si se invoca encolar, el elemento debe ser insertado en el casillero siguiente al que indica la variable.
* numElem: indica cuántos elementos posee la cola. Definiendo MAX\_ELEM como el tamaño máximo del arreglo, y por lo tanto de la cola, entonces la cola está vacía si numElem==0 y está llena si numElem==MAX\_ELEM.



Un detalle faltante es el siguiente: ¿qué pasa si la variable ultimo sobrepasa el rango de índices del arreglo? Esto se soluciona definiendo que si después de insertar un elemento el índice ultimo == MAX\_ELEM, entonces se asigna ultimo = 0 , y los siguientes elementos serán insertados al comienzo del arreglo. Esto no produce ningún efecto en la lógica de las operaciones del TDA, pues siempre se saca el elemento referenciado por el índice primero, aunque en valor absoluto primero > ultimo. Este enfoque es conocido como implementación con arreglo circular, y la forma más fácil de implementarlo es haciendo la aritmética de subíndices módulo MAX\_ELEM.



**DA Cola de Prioridad**

Una cola de prioridad es un tipo de datos abstracto que almacena un conjunto de datos que poseen una llave perteneciente a algún conjunto ordenado, y permite insertar nuevos elementos y extraer el máximo (o el mínimo, en caso de que la estructura se organice con un criterio de orden inverso).

Es frecuente interpretar los valores de las llaves como prioridades, con lo cual la estructura permite insertar elementos de prioridad cualquiera, y extraer el de mejor prioridad.

Dos formas simples de implementar colas de prioridad son:

Una lista ordenada:

* Inserción: O(n)
* Extracción de máximo: O(1)

Una lista desordenada:

* Inserción: O(1)
* Extracción de máximo: O(n)

**Heaps**

Un heap es un árbol binario de una forma especial, que permite su almacenamiento en un arreglo sin usar punteros.

Un heap tiene todos sus niveles llenos, excepto posiblemente el de más abajo, y en este último los nodos están lo más a la izquierda posible.

Ejemplo:

