**Estructuras de Datos**

# **Paralelo 1 – PAO II 2020 - 2021**

# **Práctica y Tarea TDA List**

# Introducción

En esta práctica, usted empezará a trabajar en una implementación funcional del TDA lista. Para esto, el código adjunto define este TDA a través de la interfaz List. Usando esta definición, usted debe escribir dos implementaciones de este TDA: una estática y otra dinámica.

Su **implementación** **estática**, que debe ser escrita en una clase llamada ArrayList, utilizará un arreglo cuyo tamaño debe “*crecer”* a medida que se necesite más espacio. Por esta razón, además de un atributo para controlar su capacidad máxima, la clase ArrayList debe tener un atributo para llevar control del tamaño efectivo del arreglo. Este atributo indica cuánto espacio del arreglo reservado está siendo realmente utilizado. Dicho atributo debe ser actualizado cada vez que un objeto sea añadido a o removido de la lista.

Por otro lado, su **implementación dinámica**, escrita en una clase llamada LinkedList, debe utilizar nodos que almacenen un contenido genérico. La lista dinámica no necesita ninguna estrategia de crecimiento puesto que su tamaño no es fijado al momento de crear una nueva instancia.

# El TDA Lista y la interface List

Para los propósitos de esta práctica, el TDA lista está definido en la interfaz List de la siguiente manera:

public interface List<E>

Note que la interfaz List es parametrizada por tipo (que se especifica con el nombre E en referencia a que se trata de una lista de *elementos*). Esta parametrización nos permitirá instanciar listas de elementos de cualquier tipo (p. ej., listas de números enteros, listas de estudiantes, o incluso listas de listas).

La interfaz List contiene los siguientes métodos para añadir elementos a (y removerlos de) la lista y para recuperarlos:

public boolean addFirst(E e); // inserta el elemento e al inicio

public boolean addLast(E e); // inserta el elemento e al final

public E removeFirst(); // remueve el elemento al inicio de la lista

public E removeLast(); // remueve el elemento al final de la lista

La interfaz List incluye además los siguientes métodos de inserción y eliminación de elementos en posiciones específicas:

public void add (int index, E element); // inserta element en la posición index

public E remove (int index); // remueve y retorna el elemento en la posición index

public E get (int index); // retorna el elemento ubicado en la posición index

public E set (int index, E element); // setea el element en la posición index

La interfaz también especifica los siguientes métodos, que permiten conocer el tamaño de la lista, verificar si ésta está vacía, y vaciarla.

public int size();

public boolean isEmpty();

public void clear();

Finalmente, la lista tiene un método llamado toString, que retorna una cadena de caracteres representando los elementos que la lista contiene:

public String toString();

# Lista Estática: Clase ArrayList

Como se especificó antes, esta clase es la implementación estática del TDA lista. Por tanto, internamente, debe utilizar un arreglo.

Para cumplir con la especificación del TDA, esta clase también debe ser parametrizada por tipo. Sin embargo, es importante notar que, actualmente, **Java no permite declarar arreglos con tipos parametrizados**. Por tanto, la siguiente línea de código **NO** es posible:

E elements[] = new E[capacity]; ❌

La línea anterior genera un error en tiempo de compilación: la máquina virtual indica que no es posible crear arreglos de genéricos. Por esta razón, el arreglo de elementos de la clase ArrayList debe ser declarado e inicializado de la siguiente manera:

E elements[] = (E[]) new Object[capacity]; ✔️

Note que el problema se soluciona usando la clase Object y un cast. Esta será la única ocasión que utilicemos casting en este curso ya que, como lo hemos discutido antes,este tipo de operación **no es recomendable**. Sin embargo, este *cast* está dentro de la clase ArrayList y quien use la clase no se enterará, necesariamente, de los detalles internos de implementación.

El atributo capacity controla el tamaño inicial de todo ArrayList.

Note que, en un ArrayList, las operaciones de inserción y eliminación de elementos deben modificar su tamaño efectivo. Asimismo, la clase actualizar su capacidad cada vez que un se requiera insertar un nuevo elemento en un ArrayList que está lleno. La política de crecimiento que usaremos en este curso es que un ArrayList lleno **crecerá al doble de su tamaño actual** cuando se intente añadir un nuevo elemento. Es recomendable que usted cree un método privado llamado addCapacity() para manejar esto.

# Lista Dinámica: Clase LinkedList

La clase LinkedList corresponde a la implementación dinámica del TDA lista. Esta clase, por tanto, necesita, además, de otra clase parametrizada: la clase Node. Un nodo tiene un contenido genérico, al que llamaremos content, y una referencia al nodo siguiente (el atributo next).

public class Node<E> **{**

private E content;

private Node<E> next;

// métodos

**}**

La clase LinkedList tendrá tres atributos. Los dos primeros serán referencias al primer y último nodo de la lista. En nuestro curso, estos atributos serán llamados, respectivamente, first y last. El tercer atributo registrará el tamaño de la lista. Al igual que en un ArrayList, este atributo deberá ser actualizado cada vez que un nodo sea insertado en o removido de la LinkedList. El valor de este atributo es el que se debe retornar en el método size().

# Iterando sobre los elementos de un ArrayList y una LinkedList

Dado que la clase ArrayList utiliza un arreglo, es posible iterar sobre sus elementos utilizando un índice controlado por un lazo for que inicie en 0 y termine en el valor del tamaño efectivo del ArrayList.

Los elementos de una LinkedList, sin embargo, **no tienen un índice asociado** (puesto que esta clase no utiliza un arreglo en su implementación). Por tanto, para iterar sobre una LinkedList, debemos usar una variable que *viaje* a través de sus nodos. Esta iteración puede ser hecha mediante el siguiente lazo for:

for (Node<E> n = this.first; n != null; n = n.getNext()){

// n es el nodo actual

E content = n.getContent();

}

El lazo de arriba esencialmente utiliza un *nodo viajero* n, que parte desde el primer nodo de la lista (n = this.first) y en cada iteración avanza al siguiente nodo (n = n.getNext()). Note que este lazo termina cuando el nodo viajero es igual a null.

# Probando su Código: La clase Main

Escriba un programa principal para probar el funcionamiento de su TDA. Para esto, instancie dos listas–una estática y una dinámica–de números enteros, como se muestra a continuación:

List l1 = new ArrayList<Integer> ();

List l2 = new LinkedList<Integer> ();

Añada elementos a sus listas utilizando los métodos implementados. Finalmente, utilice el método toString de ambas listas para imprimir sus contenidos.