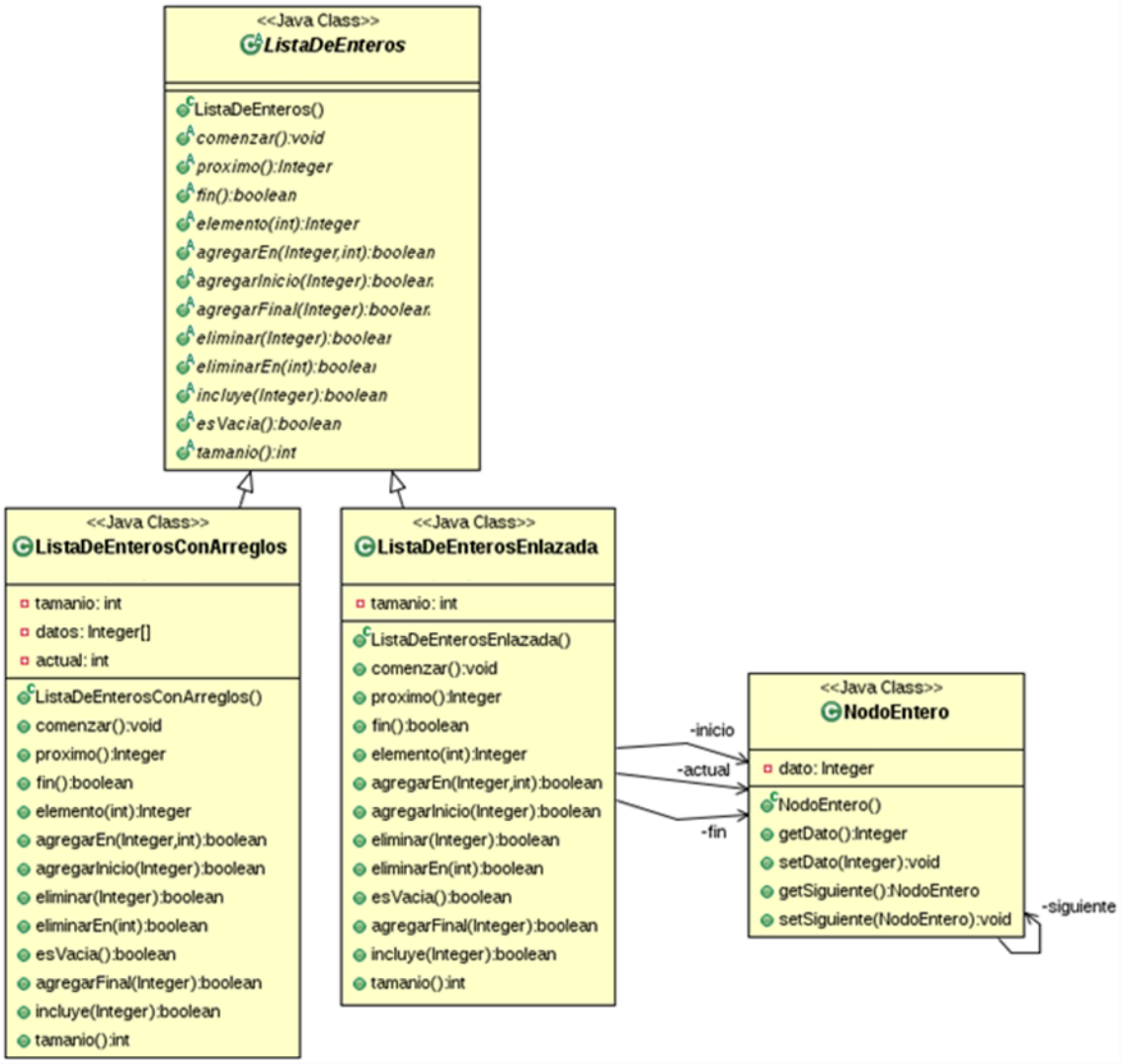
Práctica 1
Abstracción y encapsulamiento
Herencia. Polimorfismo. Tipos Genéricos

Importante: Descargue el material disponible del Campus. Se recomienda trabajar en un mismo proyecto (**AyED**) y crear un paquete por cada ejercicio.

1.Considere la siguiente especificación de operaciones de una lista de enteros



Donde:

1.1. Importe en su proyecto en Eclipse (o IDE similar) el archivo **ListasDeEnteros.zip** provisto por la cátedra usando la opción de Import correspondiente. Para poder usar las listas de enteros y sus operaciones, en cada una de las declaraciones de clases se debe agregar **import tp01.ejercicio1.*;**

1.2. Escriba una clase llamada **TestListaDeEnterosConArreglos** que reciba en su método **main** una secuencia de números, los agregue a un objeto de tipo **ListaDeEnterosConArreglos** y luego imprima los elementos de dicha lista.

1.3. Escriba una clase llamada **TestListaDeEnterosEnlazada** que reciba en su método **main** una secuencia de números, los agregue a un objeto de tipo **ListaDeEnterosEnlazada** y luego imprima los elementos de dicha lista.

1.4. ¿Qué diferencia encuentra entre las implementaciones de los puntos anteriores?

1.5. Escriba un método recursivo que imprima los elementos de una lista en sentido inverso. La lista la recibe por parámetro.

1.6. Analice las implementaciones de la clase ListaDeEnteros **y sus subclases.**

a) **¿Podría darle comportamiento a algún método de la superclase** ListaDeEnteros**? ¿Por qué la clase se define como abstracta? Note que una subclase implementa la lista usando un arreglo de tamaño fijo y la otra usando nodos enlazados.**

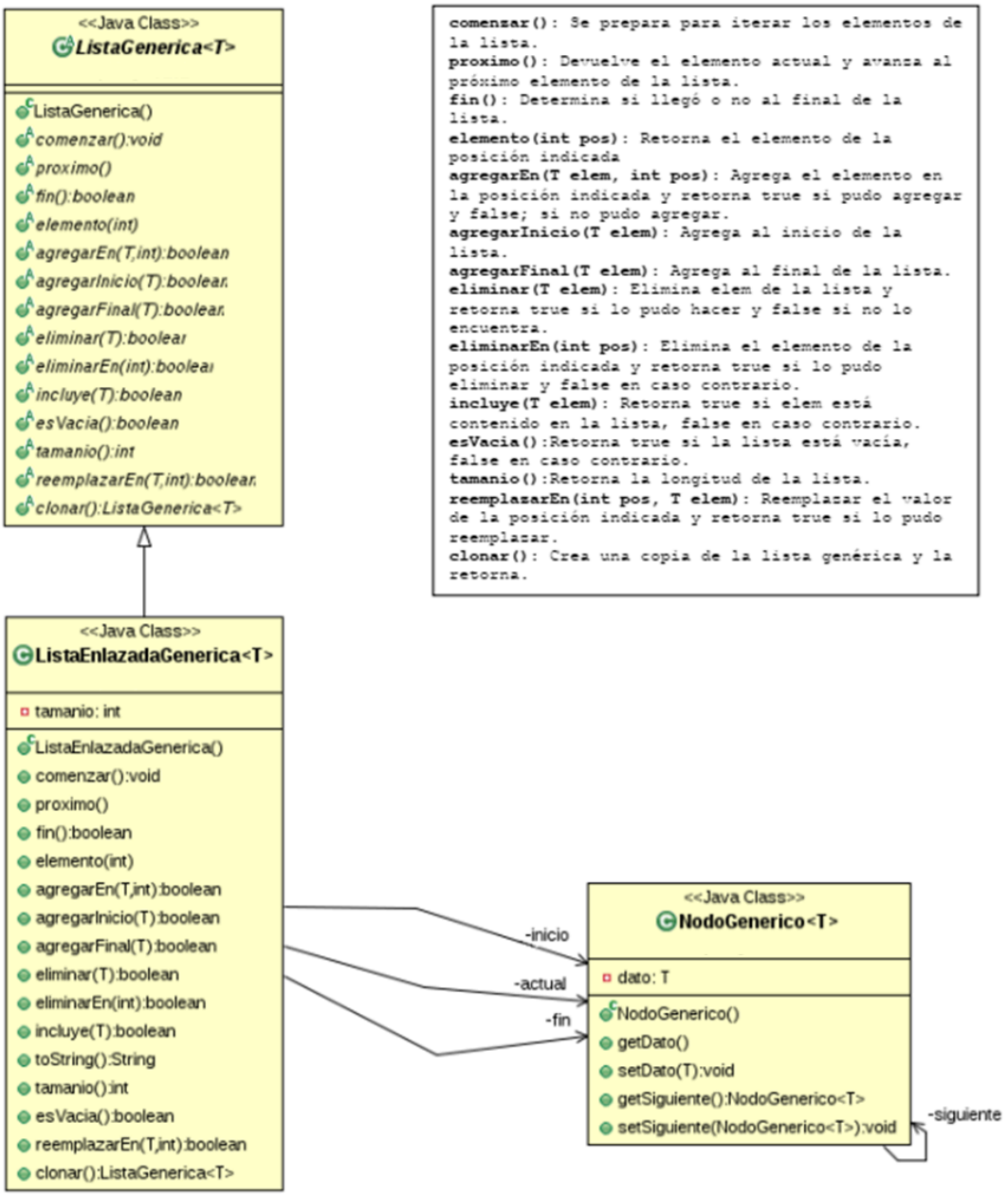
b) **Considerando los enlaces entre nodos, ¿qué diferencias existen entre agregar un nodo al principio de la lista, agregar un nodo en el medio y agregar un nodo al final?**

c) **Una lista implementada con arreglos, ¿tiene su primer elemento en el índice del vector: 0, 1 o depende de la implementación?**

comenzar(): Se prepara para iterar los elementos de la lista.
proximo(): Devuelve el elemento actual y avanza al próximo elemento de la lista.
fin(): Determina si llegó o no al final de la lista.
elemento(int pos): Retorna el elemento de la posición indicada
agregarEn(Integer elem, int pos): Agrega el elemento en la posición indicada y retorna true si pudo agregar y false; si no pudo agregar.
agregarInicio(Integer elem): Agrega al inicio de la lista.
agregarFinal(Integer elem): Agrega al final de la lista.
eliminar(Integer elem): Elimina elem de la lista y retorna true si lo pudo hacer y false si no lo encuentra.
eliminarEn(int pos): Elimina el elemento de la posición indicada y retorna true si lo pudo eliminar y false en caso contrario.
incluye(Integer elem): Retorna true si elem está contenido en la lista, false en caso contrario.
esVacia():Retorna true si la lista está vacia, false en caso contrario.
tamanio():Retorna la cantidad de elementos.

2. Tipos Genéricos

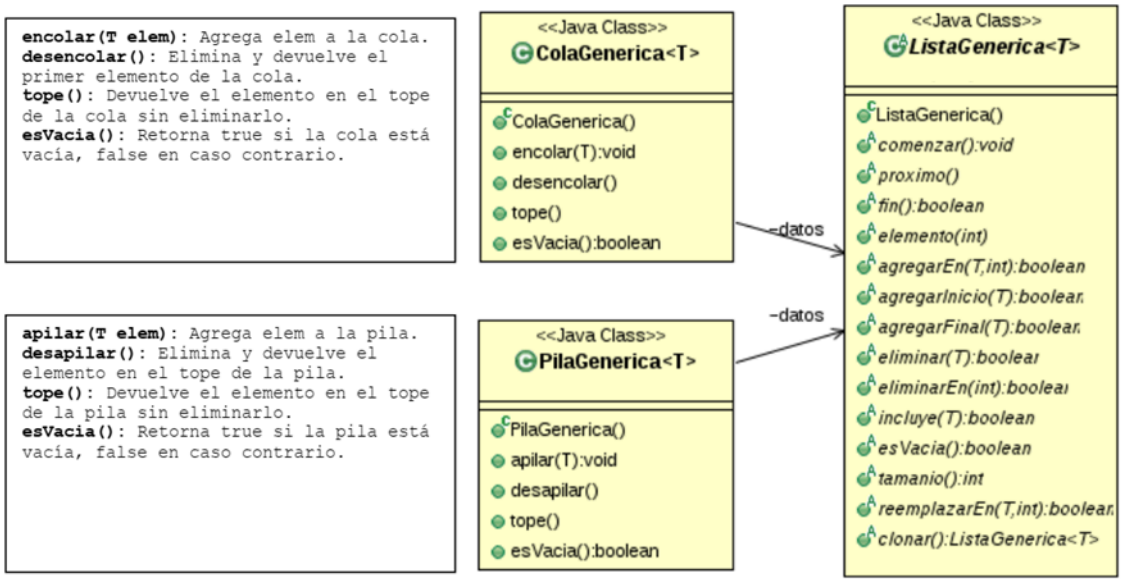
Considere la siguiente especificación de operaciones de listas genéricas:



2.1. ¿Podría resolver los ejercicios del punto 1 utilizando listas genéricas?
2.2. Importe el archivo **ListasGenericas.zip** dado por la cátedra en Eclipse (o IDE similar) usando la
opción de Import correspondiente. Para poder usar las listas genéricas y sus operaciones, en cada una de las declaraciones de clases se debe agregar **import tp01.ejercicio2.*;**
2.3. Escriba una clase llamada **TestListaEnlazadaGenerica** que cree 4 objetos de tipo **Estudiante** *(con apellido, nombre, legajo)* y los agregue a un objeto de tipo **ListaEnlazadaGenerica** usando los diferentes métodos de la lista y luego, imprima los elementos de dicha lista usando el método **tusDatos()**.
2.4. Analice las implementaciones de la clase **ListaGenerica<T>** y sus subclases, luego responda:

a) ¿Qué diferencia observa entre las implementaciones de **ListaEnlazadaGenerica** y **ListaDeEnterosEnlazada**?
b) ¿Cómo se define el nodo genérico? ¿Cómo se crea una instancia del mismo?
c) ¿Qué devuelve el método **elemento()** de la lista?
d) ¿Cómo agregaría un método nuevo? Implemente un nuevo método de la lista que se llama **agregar(T[]):boolean.** El mismo debe agregar todos los elementos del arreglo que recibe como parámetro y retornar true si todos ellos fueron agregados.

3. Sean las siguientes especificaciones de cola y pila genérica:



a) Implemente en JAVA (pase por máquina) las clases **ColaGenerica** y **PilaGenerica** de acuerdo a la especificación dada en el diagrama de clases. Defina estas clases dentro del paquete **tp01.ejercicio3**.

4. Considere un *string* de caracteres S, el cual comprende únicamente los caracteres: (,),[,],{,}.
Decimos que S está balanceado si tiene alguna de las siguientes formas:
S = "" S es el *string* de longitud cero.
S = "(T)"
S = "[T]"
S = "{T}"
S = "TU"
Donde ambos T y U son *strings* balanceados. Por ejemplo, "{( ) [( )] }" está balanceado, pero "([ )]" no lo está.
a) Indique qué estructura de datos utilizará para resolver este problema y cómo la utilizará.
b) Implemente una clase llamada **tp01.ejercicio4.TestBalanceo** (pase por máquina), cuyo objetivo es determinar si un String dado está balanceado. El String a verificar es un parámetro de entrada (no es un dato predefinido).