SESIÓN 4: Variaciones de Listas.

Pilas y Colas.

Objetivos

- Saber usar pilas y colas para resolver problemas. Evaluador o calculadora sencilla.
- Saber usar pilas y colas para resolver problemas. Resolver el problema de Josephus.
- Implementar una lista ordenada a partir de una lista enlazada simple.

Material proporcionado

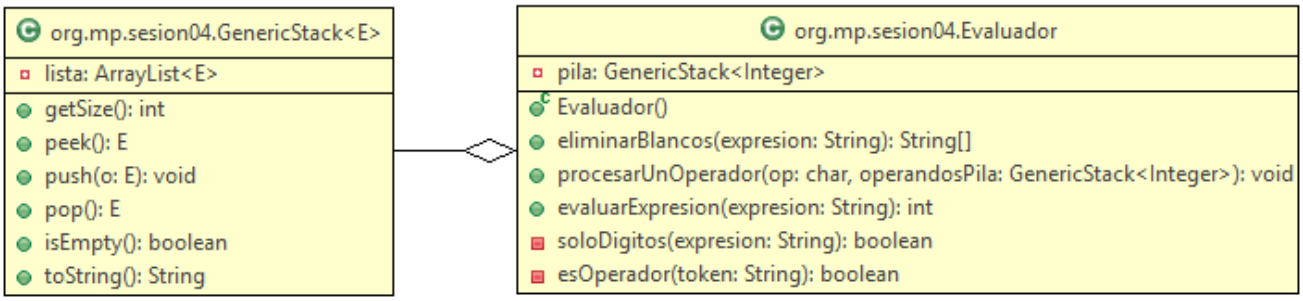
- Los test de pruebas de los distintos ejercicios propuestos, **TestEvaluador**, **TestGenericJosephus, TestSortedLinkedList.**

Requisitos

- Seguir el esquema de nombrado de paquetes, es decir: **org.mp.sesion04**. En ese paquete se crearán todos los programas que se proponen en la sesión dándoles un nombre alusivo a lo que realiza el programa y que se indica en cada ejercicio entre paréntesis y en negrita.
- Documentar todas las clases.
- Generar de todas las clases los diagramas **UML**.
- Todas las clases deberán superar con éxito los test de pruebas.
- Al final de la sesión, el alumno deberá cargar el trabajo realizado a su repositorio indicando la clave correspondiente a la sesión.
- Las sesiones se han diseñado para cubrir una semana de trabajo.

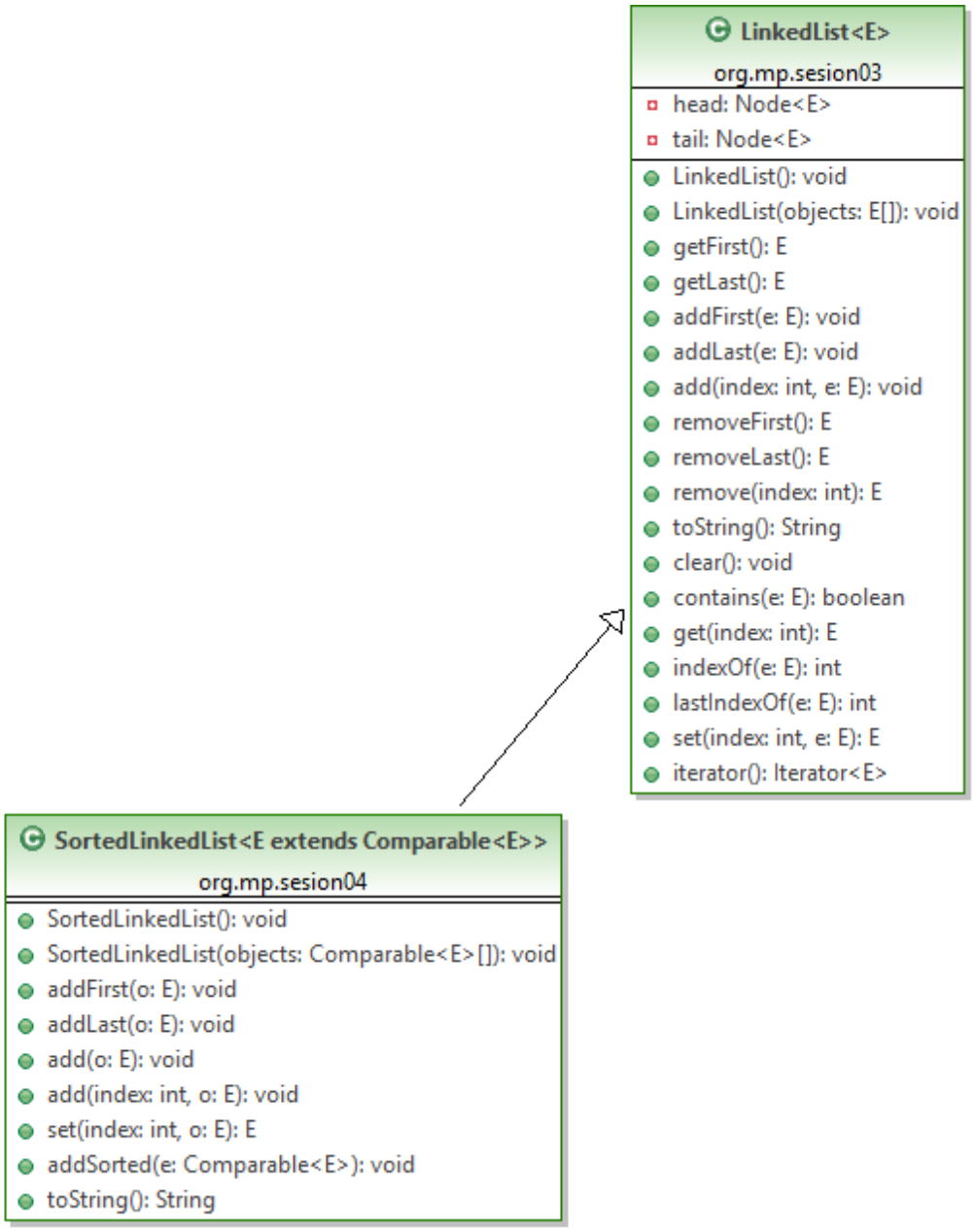
Ejercicios propuestos

1. Implementa una calculadora que permita evaluar expresiones aritméticas, con matemática entera. Por simplicidad, las operaciones que se podrán realizar son: suma, resta, producto y división. La clase se denominará **Evaluador**.
Las expresiones que evaluará la clase, se proporcionarán en *notación polaca inversa*, *notación de postfijo o notación posfija*. La expresión algebraica 5 + ((1 + 2) * 4) – 3 se traduce a la *notación polaca inversa* como 5 1 2 + 4 * + 3- y se evalúa de izquierda a derecha.
Utiliza la clase **GenericStack** estudiada en el tema 4.



2. Implementa la clase **SortedLinkedList** que permitirá trabajar con listas ordenadas. El diseño que se ha hecho para esta clase se muestra en el diagrama.

   Es una clase que hereda de la clase **LinkedList** que se ha implementado en la sesión anterior. Tiene dos constructores. Sobre-escribe algunos métodos como **addFirst, addLast, set** y **add** que *no deben estar soportados* porque su uso no garantizaría el mantenimiento del orden en la lista. Añade el método **addSorted** para conseguir una inserción en orden (orden creciente) y sobre-escribe el método **toString.**



3. Implementa la clase **GenericJosephus** que permite resolver una variante del problema de **Josephus**. Este es un problema famoso basado en una leyenda y que consiste en que **n** personas están en una situación desesperada y están de acuerdo con irse eliminando. Se disponen en un círculo (en las posiciones numeradas de 1 a n) y dado el parámetro **k**, habría que empezar a contar desde el primero hasta llegar a la posición k. El individuo que ocupa la posición k se elimina (se han saltado k – 1 individuos) y se comienza a contar de nuevo a partir del siguiente individuo hasta llegar al individuo que ocupa la posición k. Ese individuo se elimina y así continua el proceso hasta que son todos eliminados.

   Para n = 9 y k = 3
   1 2 3 4 5 6 7 8 9 se eliminarían en el orden 3 6 9 4 8 5 2 7 1