**PRÁCTICA 25 | ÁRBOLES**

# Introducción

Desgraciadamente esta práctica tomará más tiempo hacerla de lo que debería; esto se debe a que se necesita el código que convierte una expresión infija a prefija y el que convierte esto mismo a postfija. Aquella práctica fue entregada incompleta porque no me alcanzó de tiempo, y ahora toca usar lo que se debió hacer por completo ahí. El lado bueno es que tendré la oportunidad de reforzar ese tema y por fin dejar esos métodos funcionales.

Espero que después de terminar el código de la práctica incompleta no tener mucha dificultad para crear el árbol de expresión a base de la notación prefija. Está el ejemplo de código de la conversión a partir de una notación postfija; por tanto, siento que los procesos serán bastante parecidos.

# Desarrollo

Para fortuna mía resultó ser que el código que convertía una expresión infija a prefija o postfija respectivamente, funcionaba correctamente. Primero me puse a verificar el funcionamiento de lo que ya había hecho aquella vez en la práctica del tema de pilas, y después de varias pruebas correctas, pude confirmar que funcionaban los dos métodos como era debido.

Después proseguí al análisis y diseño del código para crear un árbol de expresión a partir de una notación prefija. Planteé una expresión aritmética de ejemplo y comencé a crear el árbol con el método que había pensado. Al tener creado el árbol a papel todo figuraba muy bien, entonces continué con escribir el pseudocódigo de lo que había hecho.

Posteriormente, incluí el código que se presentaba como ejemplo para la resolución del mismo problema, pero con una expresión en notación postfija. Me causó un poco de conflicto el nombre del método *infijaAArbol* en cuestión de que iba a tener dos formas de crear el árbol: convertir la expresión a prefija, o a postfija y después llamar su respectivo método privado de construcción. Pensé en crear un enum para elegir qué algoritmo usar, pero sentí que eso no le debería importar al cliente del método. Al final dejé que la propia función en su interior eligiera con un número generado aleatoriamente qué camino tomar.

Para terminar, codifiqué el pseudocódigo obtenido del análisis y diseño situándolo en la clase **ArbolExpresion** creada en una práctica anterior. Después de ejecutar el programa obtuve el mismo resultado de árbol que había creado en el análisis, además de que el recorrido en inorden del mismo era igual a la expresión ingresada, pero sin paréntesis (dato que se comentó en uno de los videos de la plataforma).

A continuación, se indican los archivos creados y modificados:

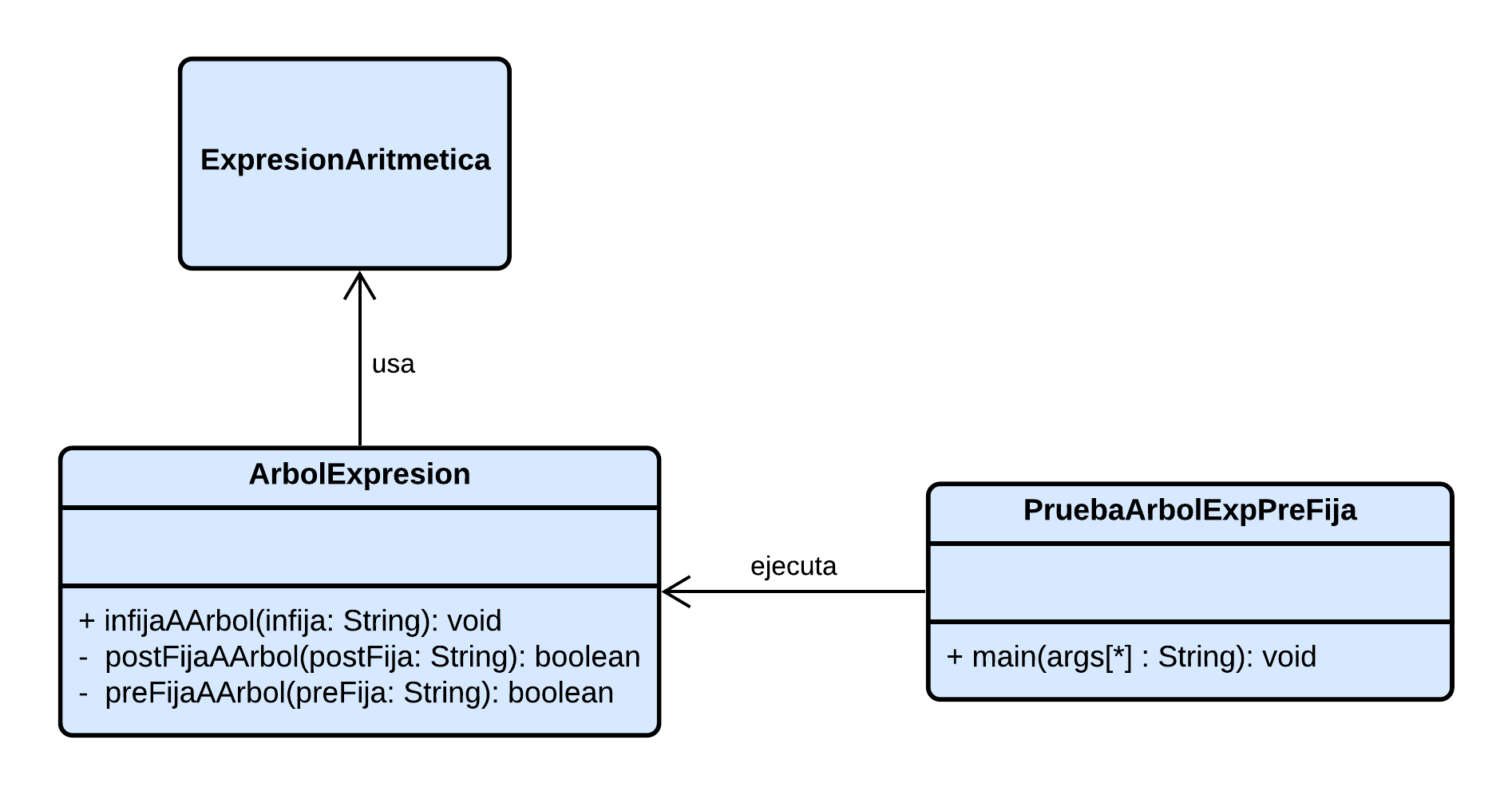
## ArbolExpresion.java (paquete ednolineal).

## PruebaArbolExpPreFija.java (paquete interfaces).

La clase anterior contiene las pruebas de esta práctica.

# Diagrama de clases

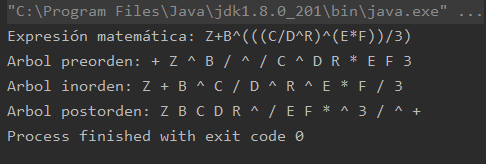
En el diagrama de clases se tiene una vista más cómoda de lo que se hizo en cada una de las clases mencionadas anteriormente. Cabe destacar que para las que ya existían en el proyecto solamente se ilustran los métodos o atributos que fueron agregados. En cambio, para las nuevas se muestra en su totalidad métodos y atributos que fueron incluidos.



**Si desea revisar la documentación de lo que fue agregado al proyecto vaya a las siguientes rutas:**

* **/docs/ednolineal/ArbolExpresion.html**
* **/docs/interfaces/PruebaArbolExpPreFija.html**

# Captura del programa funcionando



# Conclusiones

Fue bastante sencillo diseñar el método que se pedía en la práctica. Cada vez me doy más cuenta de lo útiles que son las pilas, a pesar de tener un concepto de funcionamiento tan sencillo.