**PROCESADORES DEL LENGUAJE**



**PRÁCTICA DE LABORATORIO 3**

**GRADO EN INGENIERÍA INFORMÁTICA**

Alberto Campos Paños

Rodrigo Díaz Sánchez

Leonardo Iordache

Raúl López Llana

Andrés Missiego Manjón

Lucía Picado Joglar

**[1. Prerrequisitos](#_Toc124300658)** [3](#_Toc124300658)

**[1.1. Lexer](#_Toc124300659)** [3](#_Toc124300659)

**[1.2. Parser](#_Toc124300660)** [3](#_Toc124300660)

**[2. Cómo ejecutar la práctica](#_Toc124300661)** [3](#_Toc124300661)

**[3. Estructura seguida](#_Toc124300662)** [4](#_Toc124300662)

**[3.1. Ambito](#_Toc124300663)** [4](#_Toc124300663)

**[3.2. Funcion](#_Toc124300664)** [5](#_Toc124300664)

**[3.3. Listener](#_Toc124300665)** [5](#_Toc124300665)

**[3.4. Main](#_Toc124300666)** [5](#_Toc124300666)

**[3.5. Monomio](#_Toc124300667)** [6](#_Toc124300667)

**[3.6. Polinomio](#_Toc124300668)** [6](#_Toc124300668)

**[3.7. Print](#_Toc124300669)** [8](#_Toc124300669)

**[3.8. Símbolo](#_Toc124300670)** [10](#_Toc124300670)

**[3.9. TablaSimbolo](#_Toc124300671)** [10](#_Toc124300671)

**[3.10. Valor](#_Toc124300672)** [11](#_Toc124300672)

**[3.11. Variable](#_Toc124300673)** [11](#_Toc124300673)

**[3.12. Walker](#_Toc124300674)** [12](#_Toc124300674)

**[4. Funcionalidades extra](#_Toc124300675)** [12](#_Toc124300675)

# **1. Prerrequisitos**

Para la correcta ejecución de esta práctica emplearemos el lenguaje definido en la práctica realizada anteriormente (PL2).

## **1.1. Lexer**

El léxico de nuestro lenguaje lo mantendremos igual que en la práctica anterior.

## **1.2. Parser**

El sintáctico de nuestro lenguaje ha sufrido diversos cambios:

* **Añadir línea:** para poder implementar el debugger hemos implementado línea. Esto permitirá ir línea por línea observando lo que ocurre dentro de esta.
* **Fusión de los condicionales:** hemos unido *if*, *elif* y *else* en una única palabra clave para que la estructura de este condicional funcionara correctamente.
* **Añadir return y sus respectivos parámetros:** hemos añadido el funcionamiento de la expresión *return* teniendo en cuenta que se podrán devolver los diversos parámetros que se han añadido en *parametroReturn*.
* **Cambiar asignación:** hemos cambiado el nombre de *asignación* por *declaración* y hemos denominado asignación a igualar una variable a un parámetro de asignación.
* **Añadir parámetro:** hemos creado el funcionamiento de *parámetro* para poder usarlos a la hora de llamar a una función.
* **Añadir parámetros del print y parámetros de condiciones:** hemos añadido los respectivos parámetros que podrá contener la función *print* y los que podrá contener el *condicional*.
* **Añadir operaciones y parámetros para polinomios:** hemos creado *operacionConPolinom*, *polinomioEvaluar* y *parametroPolinomio* para el correcto funcionamiento de los polinomios (tanto operar como evaluar).

# **2. Cómo ejecutar la práctica**

Para ejecutar la práctica es tan sencillo como lanzar el programa. Este te preguntará primero si deseas ejecutar un programa paso a paso o al completo. Seguidamente te pedirá la ruta del fichero que deseas ejecutar.

En caso de querer pasar algún argumento a la función main, deberemos ejecutar la aplicación pasando los argumentos en el mismo orden que queremos que se almacenen en la función main.

# **3. Estructura seguida**

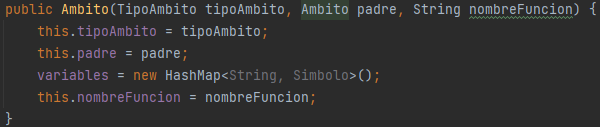
En cuanto a la estructura seguida podemos diferenciar varias partes:

* **Ambito:** es la clase encargada de almacenar los símbolos.
* **Funcion:** es la clase encargada del almacenamiento de los datos dentro del ámbito de una función.
* **Listener:** es la clase que está relacionada con el Walker. Se definirá lo que se hará en cada nodo gracias a los métodos de esta clase.
* **Main:** es la clase principal de nuestro programa.
* **Monomio:** es la clase que almacena los atributos de los monomios.
* **Polinomio:** es la clase que evalúa los polinomios y almacena los monomios para así poder operar polinomios.
* **Print**: clase encargada de recoger métodos necesarios para el correcto funcionamiento de la funcionalidad print(). Entre ellas se incluye la realización de operaciones dentro del print y la llamada a funciones.
* **Símbolo:** es la clase encargada de almacenar los nombres y valores asociados a los distintos tipos de variables y funciones que se almacenan posteriormente en los ámbitos de la tabla de símbolos.
* **TablaSimbolo:** es la clase encargada de almacenar los ámbitos y las funciones declaradas anteriormente.
* **Valor:** es la clase que se utiliza para crear objetos que se utilizarán en el resto del programa.
* **Variable:** es la clase que se usa para manejar la asignación y declaración de variables.
* **Walker:** es la clase correspondiente al Walk de ANTL.

## **3.1. Ambito**

Es la clase encargada de almacenar los símbolos para poder manejar la visibilidad de las variables principalmente. Existen 2 tipos, los ámbitos de funciones los cuales almacenan sus símbolos como si fueran un programa aislado, y los ámbitos de condicionales los cuales heredan los ámbitos padres de otros condicionales y funciones para poder hacer uso de las variables declaradas fuera de estos.

Cuando se consulta la existencia de una variable en el ámbito, primero se accede al HashMap variables correspondiente a este ámbito, si no se encuentra ninguna variable que con ese nombre se deberá buscar en el ámbito padre en caso de que exista, si no la encuentra en el padre de este, así hasta que la encuentre o en su defecto se lance una excepción del tipo “La variable a la que se está intentando acceder no ha sido declarada”.



Esta clase implementa funciones como añadir, modificar, buscar o eliminar un determinado símbolo. Además, tendrá una función para añadir un ámbito padre si este no es una función.

## **3.2. Funcion**

Es la clase encargada del almacenamiento de los datos dentro del ámbito de una función. Esta clase almacena el nombre de los parámetros y el valor que tendría una función si lo devolviera (clase Valor).

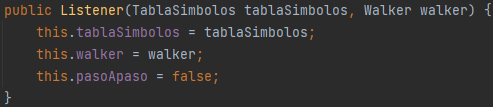
Texto

Descripción generada automáticamente

Esta clase tiene métodos que permiten obtener el nombre de la función y los nombres de los parámetros.

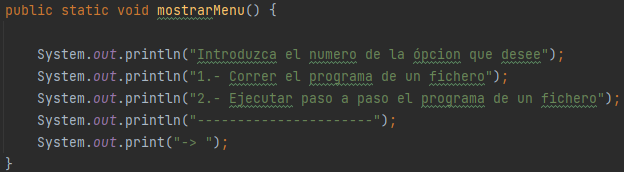
## **3.3. Listener**

Es la clase que está relacionada con el Walker. Se definirá lo que se hará en cada nodo gracias a los métodos de esta clase.



## **3.4. Main**

Es la clase principal de nuestro programa. Dentro de esta es donde se pide al usuario tanto el modo en el que quiere que sea ejecutado el programa como la ruta al fichero que quiere ejecutar.

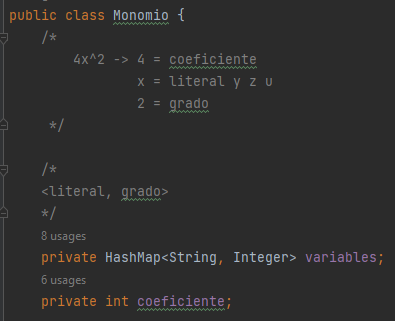


En caso de pasarle argumentos al ejecutable, es aquí mismo donde se almacenan dichos valores y los pasa como argumentos a la función main del ejecutable.

## **3.5. Monomio**

Es la clase que almacena los atributos de los monomios de un polinomio.

La clase Monomio se compone de un HashMap que guarda los literales del monomio y sus grados. También, almacenamos el coeficiente del monomio.



Tendremos dos funciones principales que nos devolverán el HashMap y el coeficiente mencionado anteriormente:

Texto

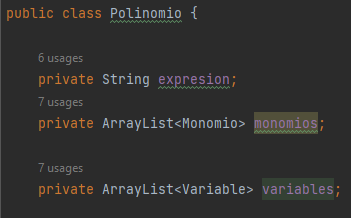
Descripción generada automáticamente 

Por último, tenemos una función que transforma el monomio a un formato deseado. Sirve para recuperar las variables y reconstruir el monomio (ej: 2x^2)

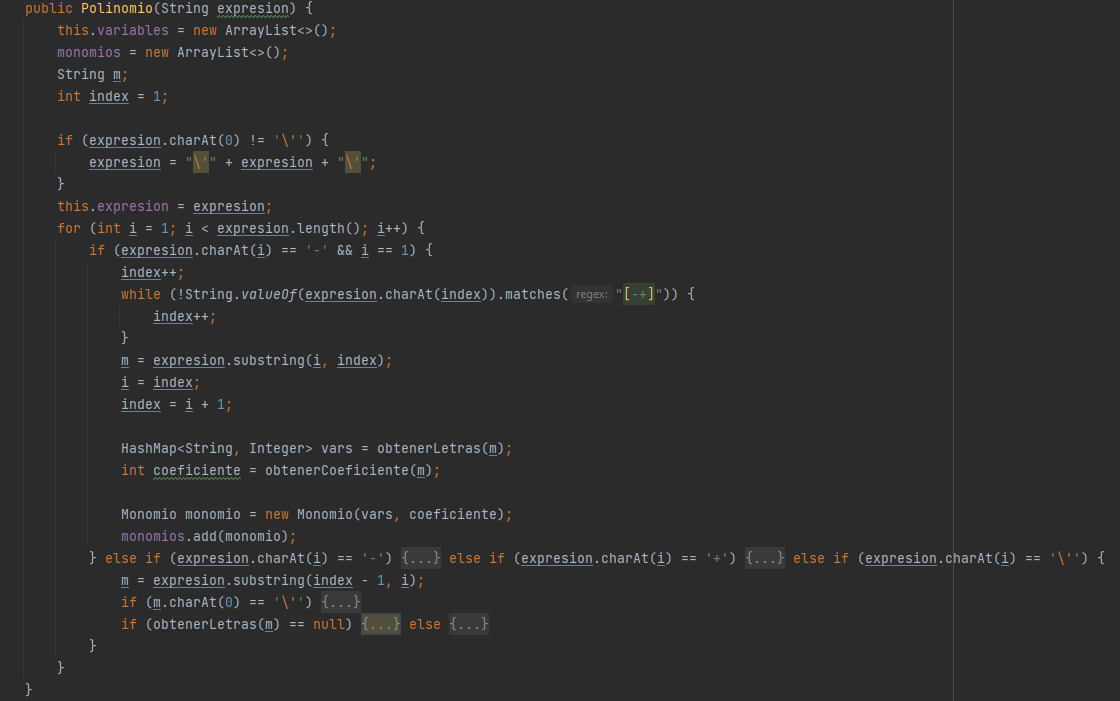
## **3.6. Polinomio**

La clase se encarga de almacenar los polinomios que se van generando a lo largo del programa. El polinomio es un String delimitado por comillas simples.

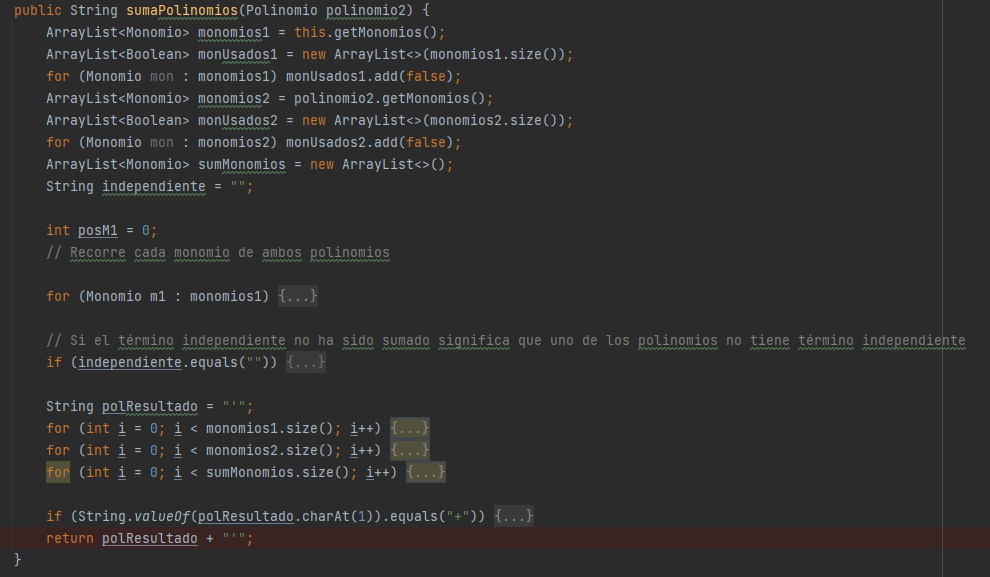
El polinomio se compone de una expresión del tipo: ‘x^2+3x+2’, la cual se almacena como un String, un ArrayList de monomios (objetos de la clase Monomio), y un ArrayList de variables.

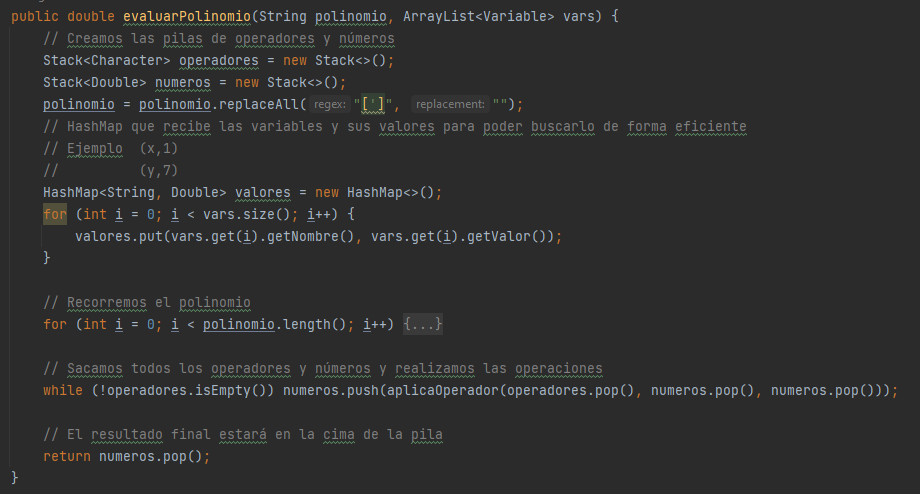


Al instanciar un objeto, el constructor separará y creará los monomios del mismo, los cuales se almacenan en el ArrayList. Este método nos permite operar monomio a monomio, facilitando las operaciones de suma y resta de polinomios.



Las funciones principales de esta clase son las siguientes:

* **sumaPolinomios:** se encarga de realizar las operaciones de suma de dos polinomios. El método realiza operaciones empleando la lista de monomios de cada polinomio y operando en aquellos con los mismos literales y grados. Para realizar la resta se hace uso de una función auxiliar, cambiaSigno (cambia el signo de todos los monomios de un polinomio) para negar el segundo polinomio y a continuación realiza la suma.
* **evaluarPolinomio:** se encarga de sustituir las variables del polinomio por los valores dados y realiza la operación, obteniendo el valor del polinomio para dichos números.



## **3.7. Print**

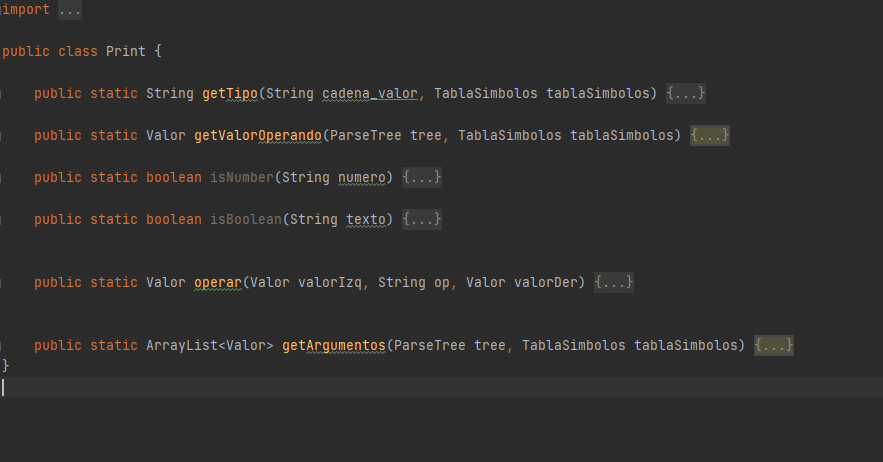
Como podemos ver, la clase Print (la cual es un contenedor de métodos que se utilizan en la funcionalidad de la instrucción print()) contiene los siguientes métodos:

* getTipo -> al cual le pasamos un string (nombre de la variable) y la tabla de símbolos y nos devuelve el tipo de la variable que se ha pasado al método junto con la tabla.
* getValorOperando -> lo utilizamos para realizar las operaciones dentro de los print. Por ejemplo, print(2+3-23), print(a + b) … Esto amplia mucho la funcionalidad básica del print. Es decir, no solo muestra por pantalla cadenas de texto, variables o números, sino que se puede realizar operaciones dentro del mismo. También trata la llamada a las funciones dentro de los print (por ejemplo, print(tres ())).
* isNumber -> para comprobar si el string que le pasamos es un número o no
* isBoolean -> para comprobar si el string que le pasamos es un booleano o no
* operar -> para realizar la operación (la operación como tal es pasada como un string) entre dos valores (trata operaciones como suma, resta... así como las operaciones and, or, etc.) este método es llamado desde getValorOperando para realizar las operaciones en forma de “tripleta”
* getArgumentos -> le pasamos el árbol y la tabla de símbolos. Sirve para obtener los argumentos de una llamada a la función.

Cabe destacar que las operaciones que se pueden realizar dentro del print son las siguientes:

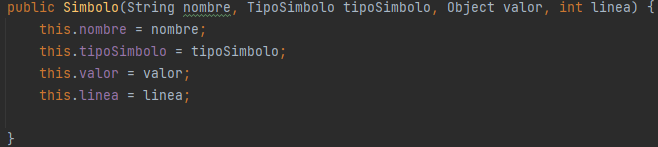
* Suma (+)
* Resta (-)
* Elevado (^)
* Multiplicación (\*)
* División (/)
* Módulo (%)
* División entera (//) -> Extra que hemos añadido
* And (&&)
* Or (||)
* XOR (##)
* Mayor que (>)
* Menor que (<)
* Mayor o igual que (>=)
* Menor o igual que (<=)
* Igual que (==)
* Distinto de (!=)

Aquí una foto de los métodos (sin mostrar el contenido de los mismos debido a su longitud en líneas de código):



## **3.8. Símbolo**

Es la clase encargada de almacenar los nombres y valores asociados a los distintos tipos de variables y funciones que se almacenan posteriormente en los ámbitos de la tabla de símbolos.



Tiene distintos métodos como *TipoSimbolo* y añadir y obtener el valor de un símbolo.

Texto

Descripción generada automáticamente

Cada tipo de dato tiene un método que devuelve una su valor en el formato correspondiente.

## **3.9. TablaSimbolo**

Es la clase encargada de almacenar los ámbitos y las funciones declaradas anteriormente.

Texto

Descripción generada automáticamente

Tiene distintos métodos como añadir y obtener funciones, añadir, obtener y eliminar un ámbito. Los ámbitos funcionan como una pila (LinkedList), por lo que solo se podrán acceder y modificar la pila mediante los métodos addAmbito(Ambito), que añade un ámbito a la cima de la pila, y el método popAmbito() devuelve el ámbito de la cima y lo elimina de la pila de ámbitos.

El HashMap de funciones alberga los nombres de las funciones con su árbol para poder ser ejecutados posteriormente cuando sean llamados.

El HashMap de paramFunciones contiene los nombres de las funciones y los nombres de sus parámetros para ejecutar correctamente las llamadas a funciones.

## **3.10. Valor**

Esta clase se utiliza para crear objetos que se utilizarán en el resto del programa. Estos objetos contendrán tanto el valor como el tipo de valor que son.

Un ejemplo de esto podría ser el valor 7 y de tipo tendría que es un “int”.

Esto se aplica para números enteros, reales, cadenas de texto y polinomios.

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

## **3.11. Variable**

Es la clase que se usa para manejar la asignación y declaración de variables.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

Tiene distintos métodos como obtener y añadir el nombre y el valor.



## **3.12. Walker**

Texto

Descripción generada automáticamenteEs una extensión del Walk de ANTLR. Esta clase nos permite hacer un overrite del método y nos facilita a la hora de entrar o no a una rama del árbol. Un ejemplo de esto podría ser una condición “if”, en caso de que no se cumpla la condición no debemos entrar en esa rama y por lo tanto en caso de no tener un overrite del método que nos permita setear con un boolean si entrar o no a la rama, el Walker por defecto entraría siempre.

**4. Funcionalidades extra**

Hemos añadido funcionalidades extra a la práctica, que se exponen a continuación y se podrán probar con el fichero de archivos “ProgTest.pdlcalc”.

Las funcionalidades extra son:

* **Bucle *Do While***
  + Realiza una primera pasada dentro del ámbito creado y este se repetirá mientras se cumpla la condición.
  + El cambiar los valores necesarios para que no se haga un bucle infinito y consiga salir de este dependerá del código definido por el programador.
  + Sigue la misma estructura que un bucle *do while* de Java.
* **Bucle *For*** 
  + Este bucle solicita 3 parámetros. Una variable ya definida o que se cree en el momento, una condición y una acción que realizará por cada una de las pasas que haga por el bucle.
  + El bucle se realizará mientras la condición introducida sea cierta.
  + Sigue la misma estructura que un bucle *for* de Java.
* **Incremento** 
  + Se aumenta en uno el valor de un integer, float o variable.
  + Se llama con el operador *“++”*
  + Ejemplo de uso: Let a = 2; a++; //El valor de ‘a’ será 1.
* **Decremento**
  + Se decrementa en uno el valor de un interger, float o variable.
  + Se llama con el operador *“--”.*
  + Ejemplo de uso: Let b = 3; a++; // El valor de ‘b’ será 2.
* **Condicional *if /Else / Elif***
  + Condicional que se ha de ejecutar siempre después de un *If,* y que solo entra en el flujo de ejecución en caso de que la condición del *If* y de los *elif* previos hayan sido falsos y su condición sea verdadera se
  + Se llama con la sentencia *“else if(condición a evaluar){Bloque de datos}”* siempre después de una sentencia if previa.