Gil Valverde, Alberto

Requisitos del programa:

El programa que se pide tomará como argumentos de línea de órdenes los nombres de cero o más ficheros de extensión .java (deberá ignorarlos en caso contrario) o leer de la entrada estándar si no se especifica ningún nombre de fichero con extensión .java en la línea de órdenes. Para cada uno de los ficheros, el programa deberá presentar:

1. La relación de clases que se definen en el mismo, indicando fila y columna del fichero en que aparece la definición.
2.
3. Para cada clase que se declare, la relación de métodos y propiedades, señalandao así mismo la línea y columna en que aparece la declaración del método o propiedad.
4.
5. La relación de identificadores que aparezcan declarados en el fichero, indicando también la fila y columna en que aparece la declaración.
6.
7. En la relación anterior deberán ignorarse los contenidos que aparezcan dentro de comentarios válidos en Java.

Función main:

- Inicialización de número de filas y columnas (variables nrow y ncol respectivamente) a 0.

- Comprobación de número de argumentos pasados por línea de órdenes (programas). Si es mayor o igual a 1 trata esos ficheros primero comprobando si su extensión es java, es decir si el nombre acaba en .java. Si acaba en .java lo trata y si no pasa al siguiente si lo hay. Si no recibe ningún argumento lee de línea de órdenes.

- Análisis:

  -Comentarios:

Una línea: “//”.* no hacemos nada porque no necesitamos contar columna porque al cambiar de línea la columna se va a resetear a 0, y el carácter salto de línea va a incrementar la columna.

Varias líneas: “/*”([^”*/”]|\n)*”*/” contamos los caracteres salto de línea y se los sumamos a nrow y sumamos a la variable ncol la longitud del string aceptado después del último \n.

-Clases:

Lo primero que nos vamos a encontrar de lo que se nos pide es una definición de una clase java (ej: public class Pepe {) por lo que buscamos cadenas las cuales contienen opcionalmente un public o private seguido de class y un nombre que empiece por $, _, Letra y que continúe por lo anterior o un número. En el caso de que encuentre alguna imprimimos -Class: <nombre>, ROW: <fila>, COL: <columna>\n incremento el valor de la columna y empieza el contexto que he denominado CLASSN.

CLASSN: Busca la llave de abrir la clase “{“ y cuando la encuentra pasa al contexto CLASS. Lo he puesto de esta manera para que la llave pueda estar en otra línea y cuente los saltos de línea correctos.

CLASS: Busca atributos de la clase, métodos o el fin de la misma. Al buscar atributos busca cadenas del tipo: opcionalmente public o protected o private separados por uno o más espacios o tabuladores, seguido de un tipo que es igual al nombre de la clase, ya que un atributo puede ser una clase y los tipos básicos cumplen la estructura sintactica de una clase separado por 0 o más espacios o tabuladores y un “;”. al encontrar uno imprime \t-Attr: <nombre>, ROW: <fila>, COL: <columna>\n e incremento el valor de columna. Para buscar métodos busco: opcionalmente su visibilidad seguido de tipo y nombre como ya hice en los atributos pero seguido de “(“ (param (“,”param)*)?”)” siendo param un tipo seguido de un nombre. Cuando encontramos una cadena que ajuste a lo descrito anteriormente actualizamos el valor de la columna y entramos en el contexto METHODN. Para buscar el fin de la clase buscamos una llave de cerrar “}” y salimos de este contexto.

METHODN: Al igual que el contexto CLASSN es usado para que busque la llave de apertura del metodo en una linea distinta. Cuando la encontramos sumamos a la variable ncol su longitud y entramos en el contexto METHOD.

METHOD: Dentro de este contexto se busca variables o su fin de la misma manera que en CLASS pero regresando al contexto CLASS. Para buscar variables buscamos {tipo} y cuando lo encontramos pasamos al contexto VAR, de esta manera se podran reconocer variables declaradas en la misma linea con el mismo tipo (ej: int a, b;).

VAR: busca {nombre} y una vez encontrado imprimimos \t\tVAR: <nombre>, ROW: <fila>, COL: <columna>\n e incrementamos la posicion de las columnas. Busca {nombre} hasta que encuentra un “;” que actualiza ncol y vuelve al contexto METHOD.+

Por último para coger todos los casos posibles añadimos [^ \t\n]+ en el cual incrementamos la columna, . que tambien hacemos lo mismo y \n el cual incrementa la fila y resetea la columna a 0.

Consideraciones:

Los valores de la fila y columna pueden diferir del editor de texto que se use ya que en mi programa empieza a contar desde la columna y la fila 0 en cambio en algunos editores como vi por ejemplo empiezan en la fila y columna 1.

El fichero de prueba que he usado es prueba.java