Práctica final: Programación de un intérprete de BASIC

TOB

Iglesias Naveiras, Jonathan 36116150D

Curso 2011/2012

Tabla de contenido

Breve descripción de la puesta en marcha y funcionamiento del programa.

Se trata de realizar un programa que permita ejercitar los recursos y técnicas aprendidas durante el curso. El problema elegido es un intérprete de BASIC: poder ejecutar programas simples.

Sobre el enunciado de la práctica, se han realizado una serie de modificaciones, como el eliminar la restricción de las cinco órdenes. Del mismo modo, se ha modificado la notación BNF de la entrada indicada en el enunciado, para soportar todas las extensiones implementadas. Dichas extensiones son la presencia de distintos tipos de datos (ENTERO, FLOTANTE, BOOLEANO, CADENA) así como la implementación de los operadores lógicos (>, >=, <, <=, ==, <>) necesarios para la implementación de la estructura de decisión IF/ELSE/ENDIF y la estructura de repetición WHILE/WEND. Por último se ha modificado también para permitir que los nombres de variable puedan contener números, si bien deben empezar por una letra.

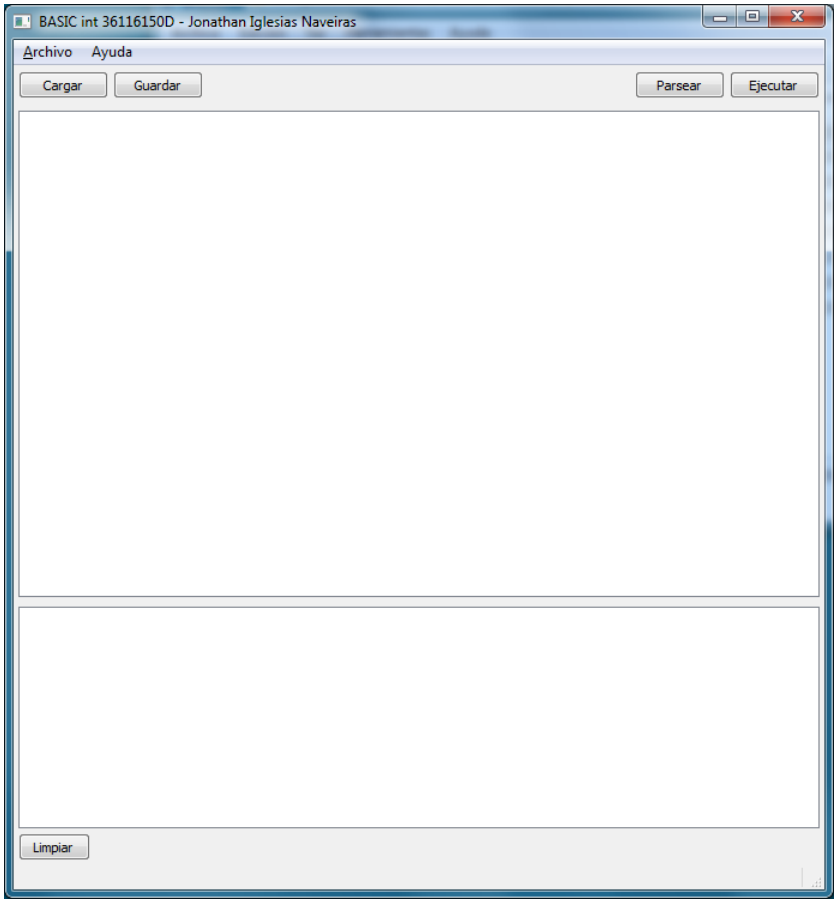
A posteori se ha implementado un entorno gráfico utilizando Qt que permite escribir y editar programas existentes, además de parsearlos y ejecutarlos.

La sintaxis aceptada por el intérprete es similar a la propuesta como ejemplo en el enunciado de la práctica. A continuación se muestra un ejemplo, utilizando también las instrucciones añadidas (programa2.bas):

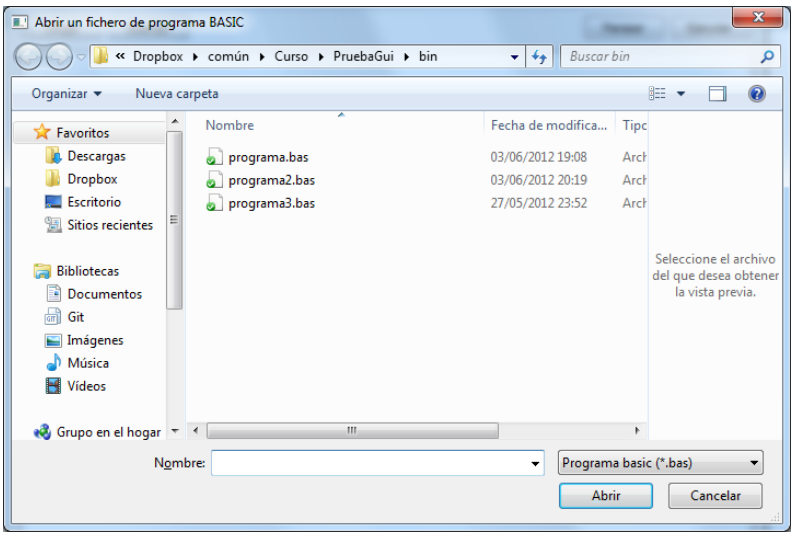
```
            REM programa de prueba
            DIM prueba as CADENA
            DIM numero as FLOTANTE
            DIM contador as ENTERO
            DIM iterador as ENTERO
            PRINT "Probando literales de tipo flotante:"
            LET numero = -3.45 + 4.56
            PRINT "Resultado: ",numero
            LET iterador = 0
            INPUT "Intruduzca valor: ",contador
            PRINT "Valor introducido: ",contador
            WHILE iterador < contador
                LET iterador = iterador + 1
                IF 3 > iterador
                    PRINT "Iteracion ",iterador,": VERDADERO: FUNCIONA EL IF"
                ELSE
                    PRINT "Iteracion ",iterador,": FALSO: FUNCIONA EL IF"
                    IF 4 < iterador
                      PRINT "Iteracion ",iterador,": VERDADERO anidado: FUNCIONA EL IF"
                    ELSE
                      PRINT "Iteracion ",iterador,": FALSO anidado: SALIENDO DEL BUCLE"
                      GOTO finbucle
                    ENDIF
                ENDIF
            WEND
:finbucle   LET prueba = "cadena"
            PRINT prueba
            LET prueba = prueba + " nueva"
            PRINT prueba
            PRINT "Resultado: ", prueba, " insercion"
```

El funcionamiento de la práctica es el siguiente:

Situados en la carpeta “bin” que contiene el ejecutable hacemos doble clic en interprete.exe



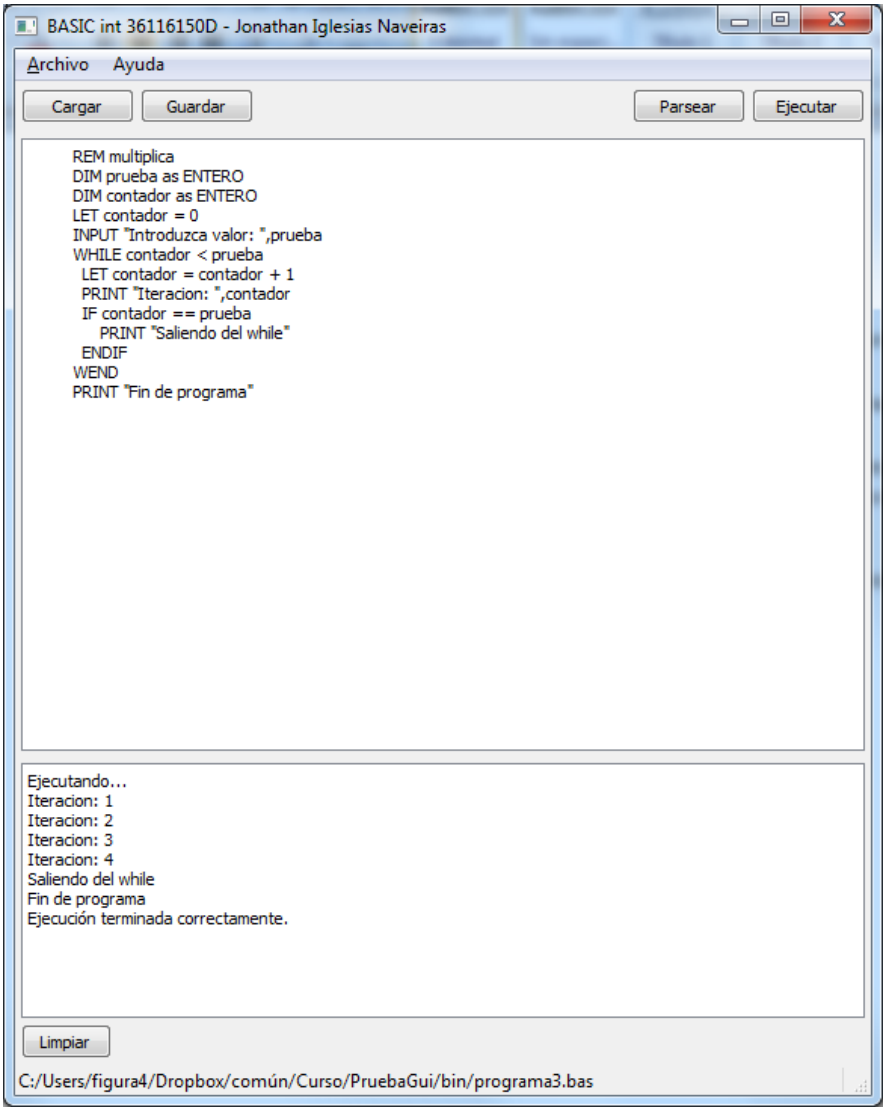
En el cuadro de texto superior es donde se podrán escribir o cargar los programas que se quieran ejecutar. Para cargar un programa se procederá haciendo clic en el botón Cargar de la esquina superior izquierda, o bien a través del menú Archivo->Cargar. En ambos casos se abrirá un diálogo de selección de fichero:



En la carpeta bin, se proporcionan 3 programas de ejemplo:

- programa.bas es el programa de ejemplo que debe ejecutar el intérprete, según el enunciado de la práctica.
- programa2.bas es el programa de ejemplo cuyo código se puede ver en la página anterior.
- programa3.bas es un programa simple que ejecuta un bucle un número de veces, mostrando por pantalla el numero de iteración, así como utilizando un if dentro del bucle para avisar en la última iteración de que se sale del bucle.

En la siguiente imagen se muestra el programa3.bas cargado, junto con su salida de ejecución:



Cuando se cargue un programa, su ruta aparecerá en la barra de estado de la aplicación.

Para la ejecución del programa cargado, es suficiente pulsar el botón Ejecutar, el cual procederá a un parseado previo, y una posterior ejecución. El botón parsear, procesa el programa, comprobando la corrección de las sentencias, generando una salida con el resultado del proceso, y deteniéndose en el momento en el que encuentra un error:

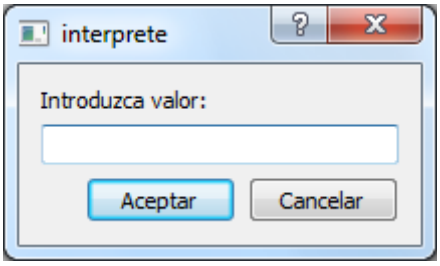
```
Parseando...
Procesando linea: REM multiplica
Procesando linea: DIM prueba as ENTERO
Procesando linea: DIM contador as ENTERO
Procesando linea: LET contador = 0
Procesando linea: INPUT "Introduzca valor: ",prueba
Procesando linea: WHILE contador < prueba
Procesando linea: LET contador = contador + 1
Procesando linea: PRINT "Iteracion: ",contador
Procesando linea: IF contador == prueba
Procesando linea: PRINT "Saliendo del while"
Procesando linea: ENDIF
Procesando linea: WEND
Procesando linea: PRINT "Fin de programa"
¡Parseado correcto!
```

```
Parseando...
Procesando linea: REM multiplica
Procesando linea: DIM prueba as ENTERO
Procesando linea: DIM contador as ENTERO
Procesando linea: LETR contador = 0
Palabra clave no reconocida: LETR
```

En el caso de que el programa cuente con alguna instrucción INPUT de petición de datos, el intérprete mostrará un cuadro de diálogo, solicitando dicha entrada. Por ejemplo, en el caso del programa ejecutado, la instrucción:

```
INPUT "Introduzca valor: ",prueba
```

Mostrará el siguiente cuadro de diálogo:



En caso de pulsar Cancelar, el programa aborta su ejecución.

Si se selecciona la opción Nuevo del menú Archivo, se podrá escribir el programa en BASIC sin necesidad de cargarlo previamente. Del mismo modo, los programas cargados pueden ser modificados y posteriormente guardados mediante el botón guardar de la esquina superior izquierda, o bien mediante el menú Archivo.

Las versiones de los recursos utilizados para el desarrollo del entorno gráfico son:

- Librerías Qt 4.8.2, compilado para generar ejecutables estáticos.
- Qt Creator 2.5.0
- minGW 4.4.0

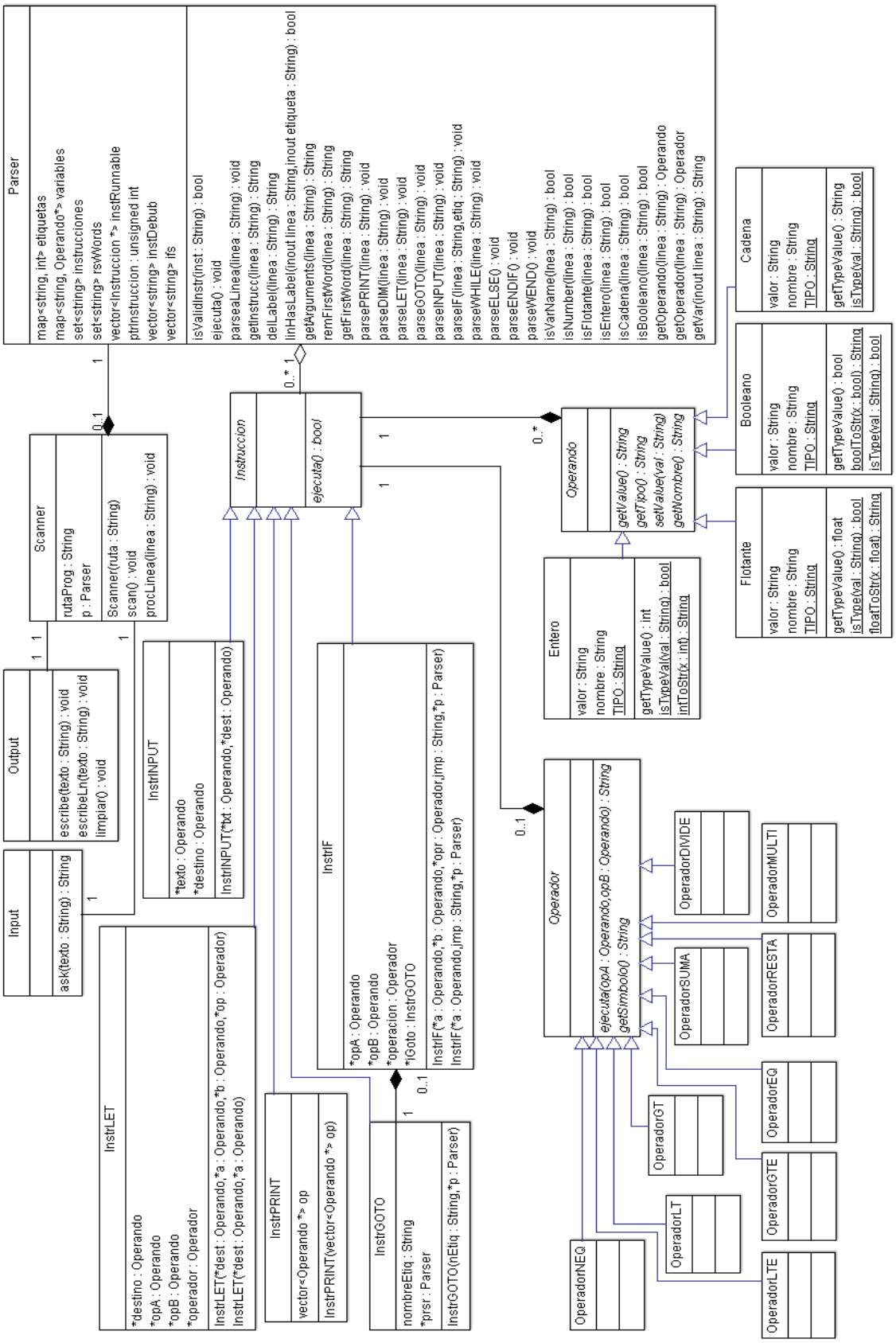
Descripción BNF de la parte específica de la sintaxis aceptada por el intérprete:

A continuación se muestra la descripción en forma BNF de la parte específica de la sintaxis aceptada por el intérprete:

```
programa ::= programa instruccion | instruccion
instruccion ::= etq: comando | comando
etq     ::= id
comando ::= REM
  | DIM id
  | LET id = expr
  | PRINT [opprint]
  | INPUT cadena, operando
  | GOTO id
  | IF exprLogica
      {comando}
    [ELSE
      {comando}]
    ENDIF
  | WHILE exprLogica
      {comando}
    WEND
  | GOTO id
opprint      ::= cadena , opprint | id , opprint | cadena | id
expr         ::= operando operador operando
exprLogica   ::= operando opLogico operando
id           ::= alfa{ alfa | num }
operador     ::= opNum | opLogico
opNum        ::= + | - | * | /
opLogico     ::= < | > | <= | >= | == | <>
operando     ::= flotante | entero | booleano | cadena
flotante     ::= [-]{num}.{num}
cadena       ::= "{ alfa | num | otros }"
entero       ::= [-]{num}
booleano     ::= TRUE | FALSE
num          ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
alfa         ::= a..z
otros        ::= resto de caracteres
```

Diagrama de clases UML.

En la siguiente imagen se muestra el diagrama de clases de la práctica:



La imagen también está adjunta en el directorio “doc” por si no se pueden puede apreciar bien la información en la imagen embebida en este documento.

Descripción de la estructura de clases utilizada.

A continuación se explica la función de cada clase:

Scanner: Se encarga de procesar el fichero de programa, comprobando que cada línea del programa comience por una instrucción correcta. En caso de que no sea correcta, lanza una excepción y sale del programa. Si la línea es correcta se procede a su parseado, proporcionándosela al parser.

Parser: El parser interpreta las líneas recibidas del scanner, creado las instrucciones necesarias para la ejecución de cada línea. Si no ha habido ningún problema en el parseado, se añade la instrucción a la lista de instrucciones y se devuelve el control al parser. En caso contrario, se lanza una excepción del tipo adecuado en función del error.

Instruccion: Es la clase abstracta que representa a cualquier comando del programa introducido. Mencionar que no todas las instrucciones soportadas por el programa tendrán su representación en forma de clase, ya que por ejemplo, la instrucción WHILE, se transforma en un IF con etiqueta, y en el momento en que se encuentre WEND, se inserta una instrucción GOTO a la etiqueta del IF, y se añade una etiqueta a la instrucción inmediatamente posterior, donde saltará el IF si no se cumple su condición. De este modo, las instrucciones del intérprete que tienen clase propia son: INPUT, PRINT, LET, GOTO e IF

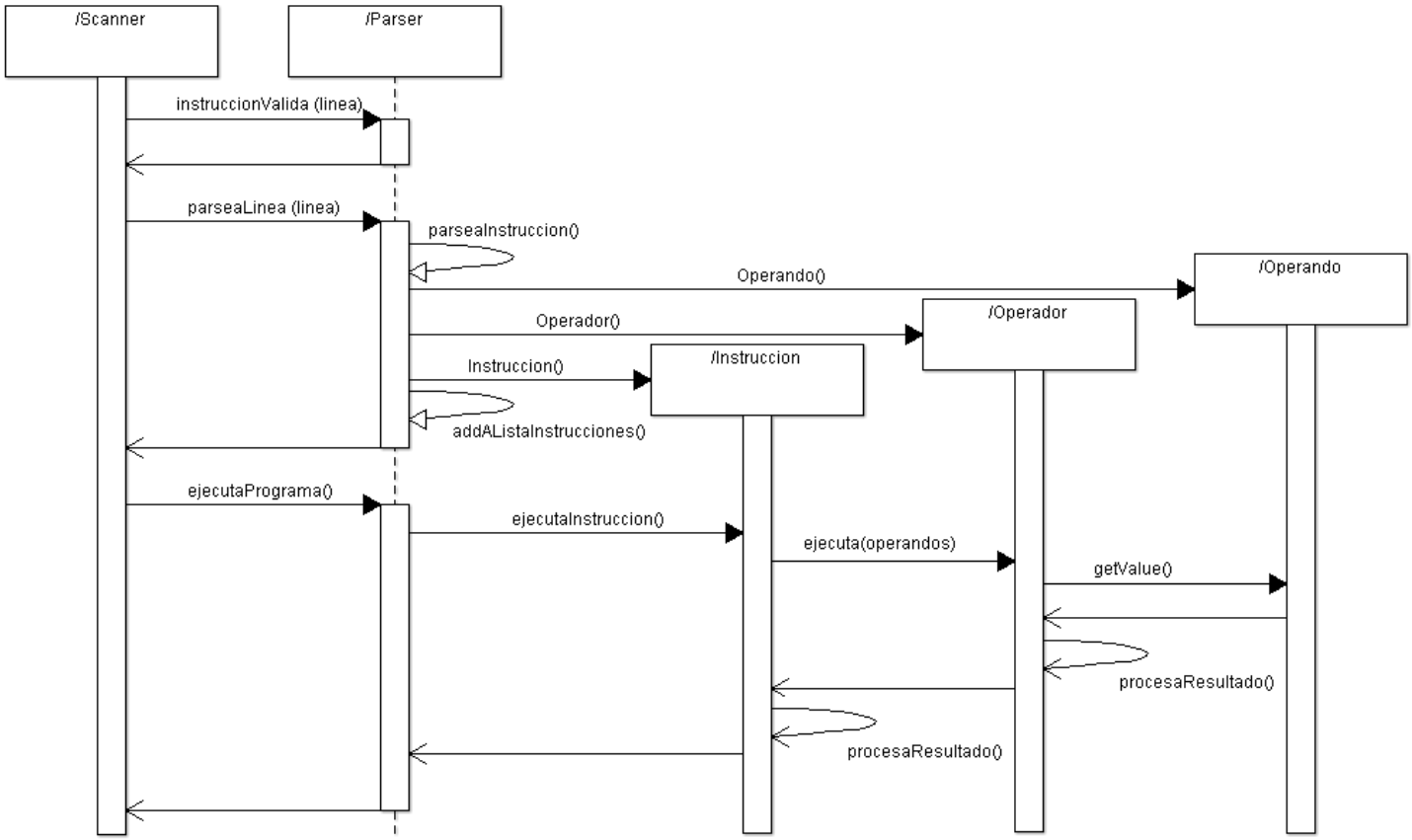
Operador: Es la clase base (abstracta) para representar las operaciones soportadas, en este caso /, *, +, -, <, <=, >, >=, ==, <>. Cada uno de estos operadores tendrá su propia clase, que se encargará de ejecutar el operador sobre los operandos proporcionados.

Operando: Es la clase base para los tipos de dato soportados por el intérprete. En este caso ENTERO, FLOTANTE, BOOLEANO y CADENA. Cada uno de estos operandos tendrá su propia clase, que se ocupará de las operaciones relacionadas con el tipo de dato en concreto.

Input: Es la clase que encapsula la petición de datos al usuario. Esta clase se hizo patente al cambiar el diseño original a través de consola, a GUI. Gracias a la implementación de esta clase, el cambiar de método de entrada sólo implicará cambios en la estructura de la misma.

Output: Es la clase que encapsula la salida de datos del programa. Al igual que la anterior, se hizo patente al cambiar el diseño original a través de consola, a GUI. Gracias a la implementación de esta clase, el cambiar de método de salida sólo implicará cambios en la estructura de la misma.

Diagrama de secuencia.



Descripción de las carpetas de la práctica.

Carpeta raíz.

\bin : Carpeta con el ejecutable.

Contiene el ejecutable de la práctica y tres programas de ejemplo. El fichero programa.bas contiene el programa de ejemplo que aparece en el enunciado de la práctica.

\doc : Documentación

Contiene la documentación relativa a la realización de la práctica. Incluye, este fichero, las imágenes del diagrama de clases y del diagrama de secuencias y el fichero del proyecto argoUML(interprete.zargo)

\doc\Documentación Doxygen : Documentación generada por Doxygen

\src : Código fuente de la práctica, distribuido en subcarpetas.