Universidad Técnica Nacional

Primer Proyecto Programado

Programación II

Lenguaje de Programación GiraCODE

Elaborado por: Mario Acuña Paniagua

2014

Tabla de contenido

[Introducción 2](#_Toc401771399)

[Desarrollo 3](#_Toc401771400)

[Descripción del Problema 3](#_Toc401771401)

[Diagrama de Clases 4](#_Toc401771402)

[Manual de Usuario 4](#_Toc401771403)

[1. Interfaz Grafica 5](#_Toc401771404)

[2. Sintaxis 6](#_Toc401771405)

[Discusión de resultados 9](#_Toc401771406)

[Conclusión 10](#_Toc401771407)

[Sugerencias y Trabajos futuros 11](#_Toc401771408)

[Referencias Bibliográficas 12](#_Toc401771409)

[Anexos 13](#_Toc401771410)

[Diccionario de Palabras reservadas 13](#_Toc401771411)

[Primer Avance del proyecto 14](#_Toc401771412)

[Prototipo de interfaz 14](#_Toc401771413)

[Diagrama de clases 14](#_Toc401771414)

# Introducción

Con el paso del tiempo, la programación se ha convertido en una herramienta indispensable para casi todos los usuarios de las computadoras, a tal punto que muchas investigaciones demuestran que en un futuro, el conocimiento de algún lenguaje de programación deberá ser una obligación y no una opción como hoy en día. Por esta razón, se ha planteado el desarrollo de un lenguaje de programación fácil de utilizar para los usuarios, principalmente para todos aquellos de habla hispana.

En el siguiente documento, se va a desarrollar una breve descripción de las características de los lenguajes actuales y los contras que presentan hacia cierta parte de la población. También se presentara una solución, a la cual se le ha llamado GiraCODE, mediante un diagrama de clases que describe su estructura, así como también un manual de usuario para este nuevo lenguaje.

# Desarrollo

## Descripción del Problema

Con el fin de aplicar los conocimientos adquiridos en la creación y uso de listas simples, dobles y circulares, se ha desarrollado un lenguaje de programación de alto nivel, el cual pretende convertirse en una herramienta de programación, mucho más fácil de utilizar y orientada hacia la mayoría de usuarios.

Además, después de analizar varios lenguajes de programación, se ha observado que la mayoría de ellos han sido desarrollados con una sintaxis muy similar al idioma inglés, por lo que la mayoría de usuarios que deseen crear sus propias aplicaciones y que no dominen esta lengua, tendrán una barrer más, e incluso otro motivo, para no querer desarrollar su idea.

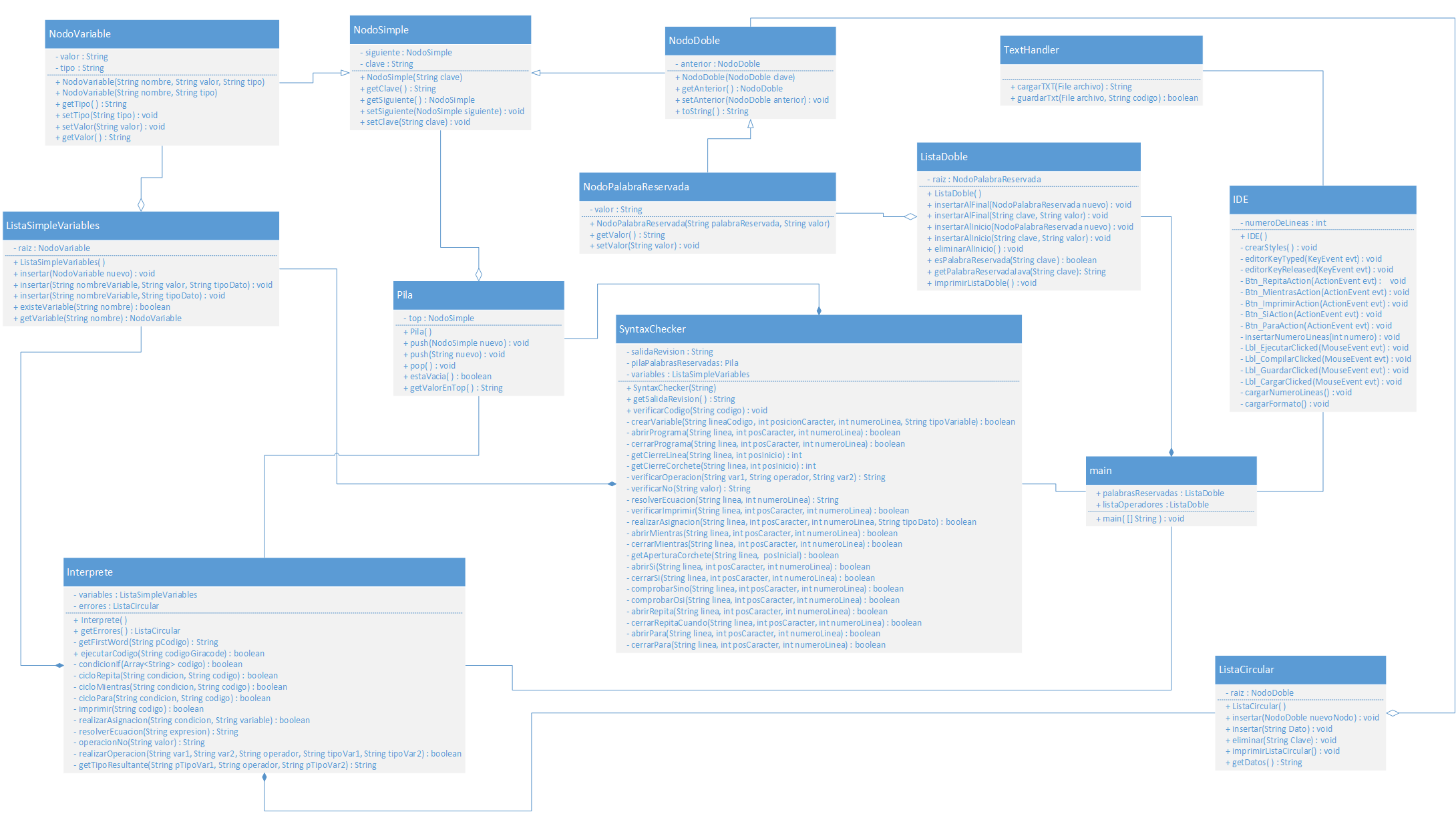
Por esta razón, con el fin de facilitar el entendimiento de la mayoría de usuarios de habla hispana, se ha pensado en desarrollar un lenguaje de programación que cuente con una sintaxis basada en la lengua española, la cual será ingresada en un editor de texto similar a un procesador de palabras, con una interfaz gráfica muy intuitiva para los usuarios.

En general, el lenguaje de programación, al cual ha sido llamado GiraCODE, cuenta con una serie de palabras reservadas, las cuales han sido almacenadas en un lista doblemente enlazada; una lista circular, encargada de almacenar cada uno de los datos que se han procesado y se mostraran en la salida, un verificador de sintaxis, basado en pilas, ciclos y condicionales, con el único fin de garantizar que el programa fuente este correctamente escrito y un módulo para ejecutar el código.

Así mismo, uno de los puntos que empaña el interés de desarrollar un programa sencillo o incluso un programa un poco más avanzado, es el tiempo empleado en el desarrollo de este. Ante este punto negativo, GiraCODE ha incorporado bloques de código preestablecidos, tales como sentencias y ciclos, los cuales facilitan la escritura de código fuente para los usuarios, agilizando el tiempo invertido por ellos.

## Diagrama de Clases

A continuación se muestra el diagrama de clases que describe la estructura del proyecto GiraCODE.



## Manual de Usuario

Para facilitar la interacción con el usuario de GiraCODE, a continuación se explica de manera detallada como utilizar el programa y algunas de las opciones que este brinda, para agilizar el tiempo de programación.

### Interfaz Grafica

Una vez ejecutada la aplicación, el usuario visualizará una ventana con varios botones, tal y como se muestra a continuación:

Ejecutar

Revisar código

Cargar código

Insertar condición Si

Guardar

Logo

Insertar sentencia imprimir

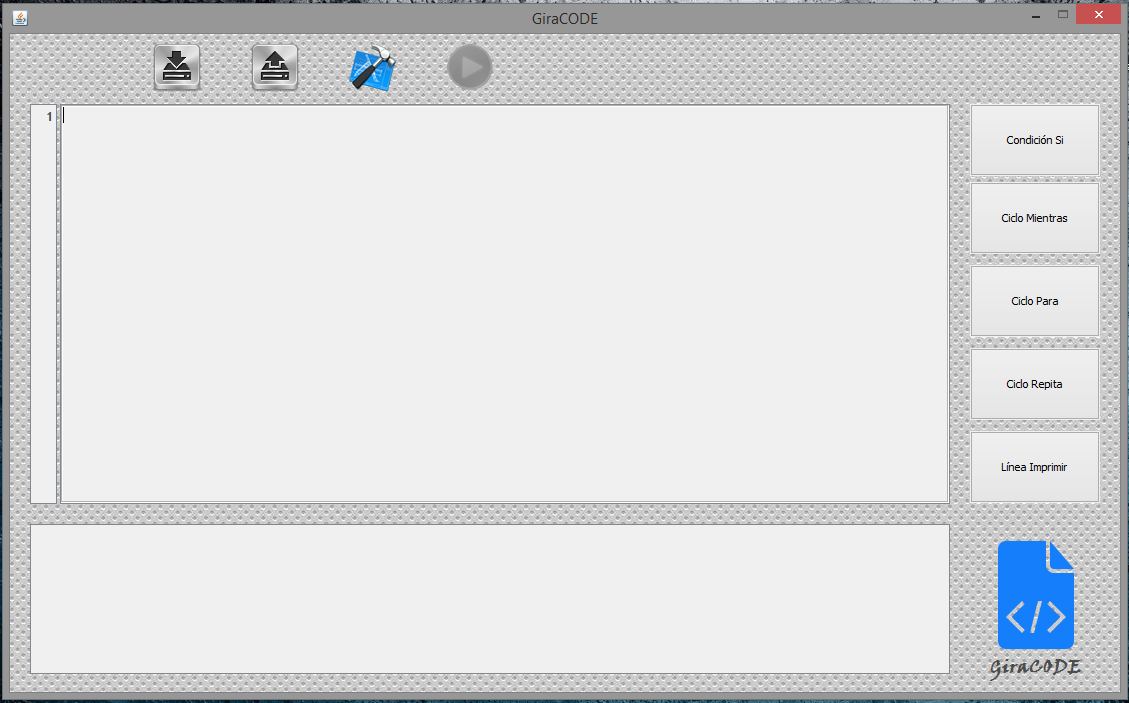
Insertar bloque repita-cuando

Insertar bloque para

Insertar bloque mientras

Edito de GiraCODE

Número de línea



Salida

**Guardar**: Permite guardar el código creado en un archivo de texto con formato txt.

**Cargar código**: permite cargar un código GiraCODE creado previamente y almacenado en un archivo de texto.

**Revisar código**: Revisa la sintaxis del código e indica si existen errores o si está escrito correctamente. En el caso de estar bien, habilita el botón para ejecutar el programa.

**Ejecutar**: ejecuta el código escrito y muestra el resultado en el espacio correspondiente. Se habilita únicamente después de verificar la sintaxis que esta correcta.

**Insertar condición si**: permite insertar el código del condicional si, con tan solo presionar el botón. El código es insertado en la posición donde se ubique el cursor en ese momento.

**Insertar bloque mientras**: permite insertar el código del ciclo mientras, con tan solo presionar el botón. El código es insertado en la posición donde se ubique el cursor en ese momento.

**Insertar bloque para**: permite insertar el código del ciclo para, con tan solo presionar el botón. El código es insertado en la posición donde se ubique el cursor en ese momento.

**Insertar bloque repita-cuando**: permite insertar el código del ciclo repita-cuando, con tan solo presionar el botón. El código es insertado en la posición donde se ubique el cursor en ese momento.

**Insertar sentencia imprimir**: permite insertar el código de la sentencia imprimir, con tan solo presionar el botón. El código es insertado en la posición donde se ubique el cursor en ese momento.

**Numero de línea**: indica el número de línea de cada sentencia. Con forme se van agregando líneas de código, se van agregando el respectivo número. Es útil para la corrección de los errores encontrados al verificar la sintaxis.

**Editor GiraCODE**: Aquí es donde se inserta todo el código deseado.

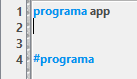
**Salida**: Aquí se muestran los errores de sintaxis del código (en caso de que el proyecto contenga errores a la hora de hacer la verificación) y el resultado de la ejecución de un programa.

**Logo**: No hace nada.

### Sintaxis

A continuación se va a explicar la sintaxis utilizada para poder desarrollar aplicaciones en este nuevo lenguaje de programación, GiraCODE:

* Inicio y cierre de programa: toda aplicación debe de iniciar con la palabra reservada ***‘programa*** nombre’ y finalizar con ***‘#programa***’.



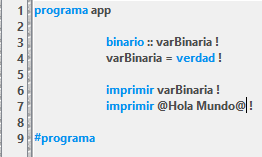
* Declaración de variables: las variables deben ser declaradas en dos secciones, la primera consiste definir el tipo (binario, entero, decimal, carácter y Texto) y nombre de las variables y la segunda consiste en la asignación del valor.

Estos pasos deben hacerse en dos líneas separadas y cada línea debe finalizar con el carácter ‘!’. Además, el nombre de las variables debe comenzar con una letra y puede contener únicamente números y letras.

La sintaxis para declarar variables es la siguiente:



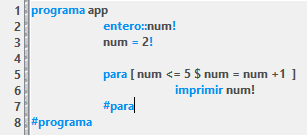
* Imprimir: Se utiliza para mostrar resultados en el cuadro de salida. Se puede imprimir el valor que contiene una variable o se puede indicar exactamente que desea imprimir. Esta sentencia debe realizarse sobre una sola línea y finalizarla con el carácter ‘!’. Aquí se muestra un ejemplo de la utilización de la sentencia imprimir:



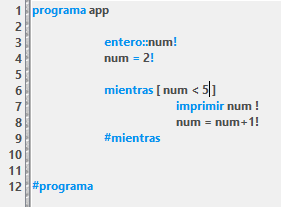
* Condicional Si: puede aparecer acompañada de ‘***osi***’ y/o ‘***sino***’. Para poder utilizarla, debe haber creado un ‘***programa’*** y tener una expresión que desee evaluar. Si al evaluar la expresión, esta no es verdad, puede realizar otra evaluación con la sentencia ‘***osi***’ o puede realizar una acción en caso que la condición se falso con el uso del ‘***sino***’. Para el ‘***si***’ y ‘***osi***’ debe de colocarse la condición entre corchetes. Por último, el ‘***si***’ debe cerrarse con la palabra reservada ***#si***. Un ejemplo del uso del condicional si, es el siguiente:



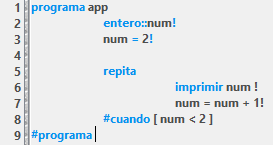
* Ciclo para: esta sentencia, ejecuta un bloque de código mientras se cumpla una condición. Para la sintaxis del ciclo ***para***, debe de agregarse una condición y una expresión para modificar el elemento condicional (separadas por el símbolo ***$***), esto se escribe entre corchetes de la forma ***para [condición $ expresión]*** y dentro del ciclo, todo aquel código que desea realizarse. El ciclo ***para***, debe cerrarse con ***#para***. Un ejemplo del ciclo es:



* Ciclo mientras: Este bloque, ejecuta el código repetidamente siempre y cuando se cumpla una condición. Para la sintaxis del ciclo ***mientras***, debe de agregarse una condición entre corchetes de la forma ***mientras [condición]*** y dentro del ciclo, todo aquel código que desea realizarse siempre que la condición se cumpla. El ciclo ***mientras***, debe cerrarse con ***#mientras***. Es importante dentro del ciclo, modificar el valor que se está evaluando, ya que de otra manera se va a enciclar el programa y nunca se podrá obtener una salida. En ejemplo es la siguiente:



* Ciclo repita-cuando: ciclo utilizado para ejecutar repetidamente un bloque de código. A diferencia del ***mientras*** y el ***para***, este ciclo ejecuta al menos una vez el código, ya que primero ejecuta y luego verifica que la condición se cumpla. Para abrir el ciclo, se escribe ***repita*** y para cerrarlo se utiliza cuando junto con la expresión entre corchetes de la forma ***cuando [expresión]***. Es importante modificar la expresión que se va a evaluar en la condición para que el programa se salga del ciclo en algún momento. Un ejemplo del ciclo es la siguiente:



En el apartado Anexos se especifican las palabras reservadas y la sintaxis comparada con el lenguaje de programación JAVA.

# Discusión de resultados

Se ha logrado finalizar el proyecto de crear un lenguaje de programación, el cual cumple con todos los requisitos solicitados en un inicio para el profesor.

Además de esto, se han agregado algunas funcionalidades extras, como por ejemplo: cargar y guardar códigos GiraCODE por medio de documentos de texto, indicar el número de las líneas y algunos botones con bloques de código. Estas características extras, permiten a los programadores conservar sus programas y agilizar el tiempo de desarrollo de aplicaciones.

Por último, para darle un aspecto más elegante al editor, se ha implementado el uso de un color (color celeste) distinto, para indicar a los programadores todas aquellas palabras reservadas por el lenguaje de programación. De esta manera, se indica a los usuarios que no es posible utilizar estas palabras para nombrar variables y se visualiza de una forma más agradable el código.

# Conclusión

Mediante este trabajo, se logró desarrollar un lenguaje de programación, el cual viene a ser una buena alternativa para todos aquellos que quieran incursionarse en el mundo de la programación. Con esta aplicación, se logra abarcar un mayor número de programadores, para los cuales el idioma se convertía en una barrera, ya que muchos no dominan el idioma inglés.

Además, se presentó la estructura del trabajo así como la sintaxis para utilizarlo y todos los elementos que contienen la aplicación.

La herramienta ya está disponible, es hora de ejecutarla y comenzar a desarrollar aplicaciones GiraCODE.

# Sugerencias y Trabajos futuros

Para los futuros trabajos programados, es importante especificar de forma más detallada los aspectos requeridos, ya que en el caso de este proyecto, solo había tres párrafos indicando el producto esperado por parte del profesor. Este proyecto era muy amplio, por lo que sería una buena opción solicitar los requerimientos del trabajo de manera más detallada por parte del profesor y no dejarlo tanto al criterio del estudiante, debido a que este último puede asumir muchos puntos y dejar de lado algunos otros.

Otra sugerencia es solicitar avances del proyecto, guiándose en la especificación de este, y no cambiarlo conforme avanza el tiempo. La razón principal de este punto es porque los desarrolladores pueden haber realizado ya los avances respectivos, y al cambiarlos, es un retroceso y se traduce en mayor tiempo invertido en el proyecto, el cual no se ve reflejado en la fecha de entrega (la fecha no fue cambiada).

Por último, sería muy útil que, a la hora de presentar los avances, el profesor brinde ideas a los estudiantes de los puntos que, a criterio de él, deban mejorarse y de igual forma, aportar críticas constructivas para obtener un mejor producto final.

# Referencias Bibliográficas

Cadenhead, R. (2002). Using Special Characters in Strings | Using Java Strings to Communicate | InformIT Recuperado el 20 de Octubre del 2014, de http://www.informit.com/articles/article.aspx?p=30241&seqNum=3

Oracle (s. f.). How to Use Editor Panes and Text Panes (The Java? Tutorials > Creating a GUI With JFC/Swing > Using Swing Components) Recuperado el 07 de Octubre del 2014, de http://docs.oracle.com/javase/tutorial/uiswing/components/editorpane.html

Oracle (s. f.). Java Platform SE 7 Recuperado el 09 de Octubre del 2014, de http://docs.oracle.com/javase/7/docs/api/

Oracle (s. f.). Swing JFC « Java Recuperado el 11 de Octubre del 2014, de http://www.java2s.com/Code/Java/Swing-JFC/CatalogSwing-JFC.htm

Pérez, B. (2014). Tupera (Versión Alnitak) [Software]. San José, Costa Rica.

Setting the Font and Color of Text in a JTextPane Using Styles | Example Depot (s. f.). Recuperado el 20 de Octubre del 2014, de http://exampledepot.8waytrips.com/egs/javax.swing.text/style\_HiliteWords2.html

Tutorial de UML - Modelo de Clases (s. f.). Recuperado el 06 de Octubre del 2014, de http://users.dcc.uchile.cl/~psalinas/uml/modelo.html

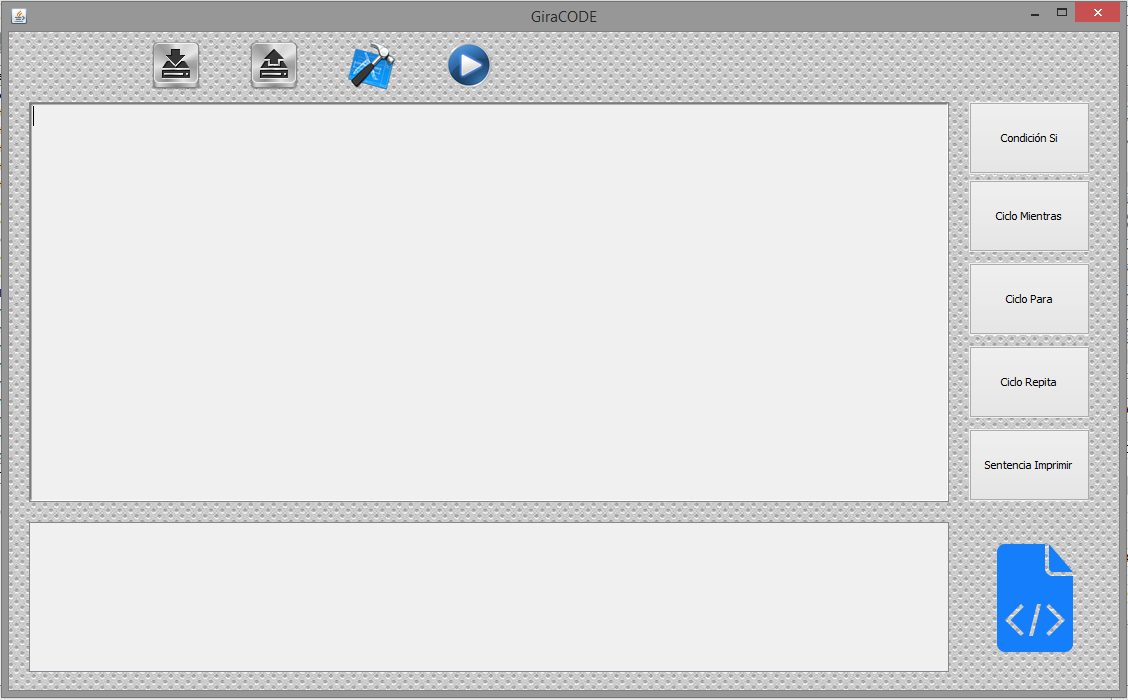
# Anexos

## Diccionario de Palabras reservadas

|  |  |
| --- | --- |
| **Lenguaje Java** | **Lenguaje GiraCODE** |
| String | Texto |
| Int | entero |
| double | decimal |
| boolean | binario |
| char | caracter |
| ; | ! |
| System.out.print | Imprimir |
| While( ){  } | mientras[ condicion ]  #mientras |
| for( ; ; ){  } | para[ condicion $ asignacion ]  #para |
| if( ){  }else if ( ){  }else{  } | si[ condicion ]  osi[ condicion ]  sino  #si |
| do{  }while( ); | repita  #cuando[ condicion ] |
| class nombre {  } | programa  #programa |
| true | verdad |
| false | falso |
| || | o |
| && | y |
| ! boolean | no[binario] |
| == | = |
| != | <> |
| Variables  tipo nombreVariable = valor; | Variables  tipo::nombreVariable!  nombreVariable = valor! |

## Primer Avance del proyecto

### Prototipo de interfaz



### Diagrama de clases

