INGENIERÍA INFORMÁTICA

Curso Académico 2006/2007

Proyecto Fin de Carrera

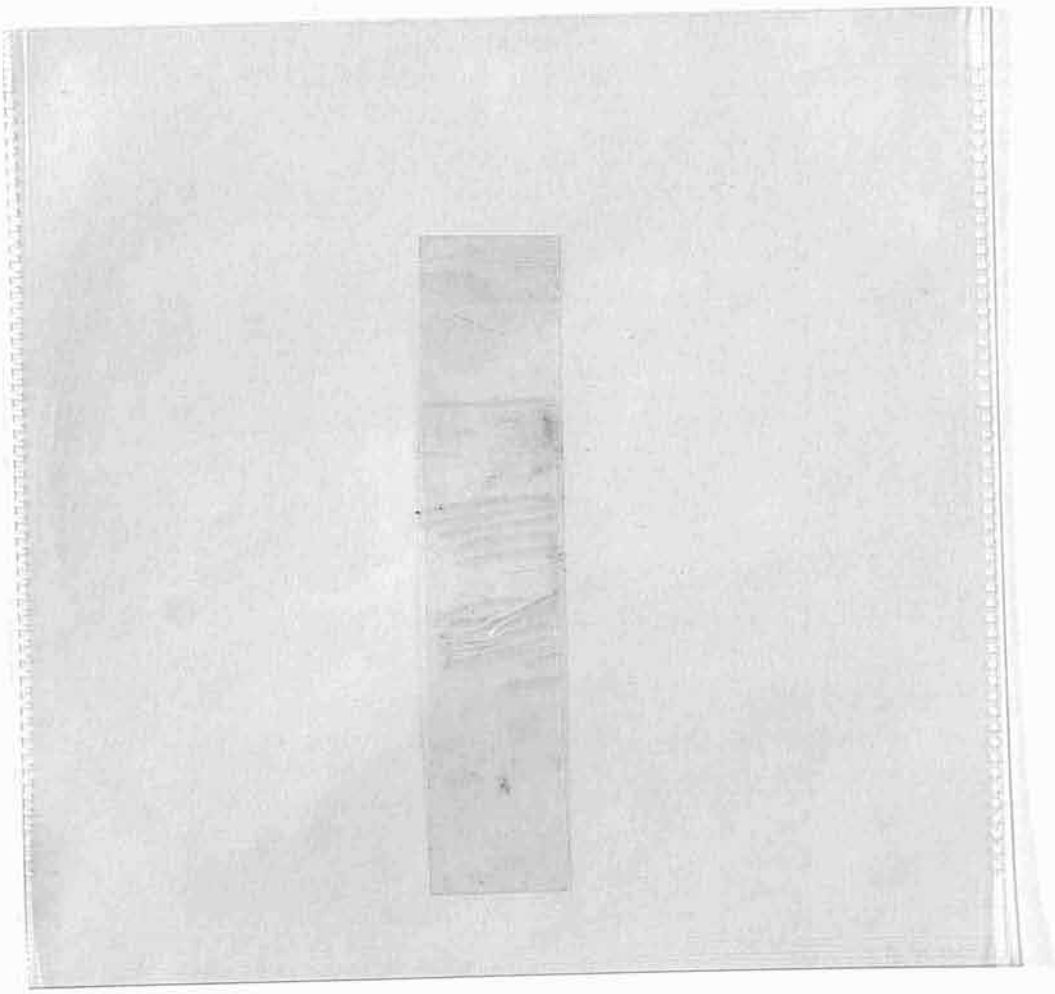
PHOSPROCESSOR

Un procesador de textos avanzado

Tutores: Micael Gallego Carrillo y

Patxi Gortázar Bellas

Autor: Adriano Galán Largo



UNIVERSIDAD REY JUAN CARLOS

INGENIERÍA INFORMÁTICA

Curso Académico 2006/2007

Proyecto Fin de Carrera

PLUSPROCESSOR

Un procesador de textos avanzado

Tutores: Micael Gallego Carrillo y

Patxi Gortázar Bellas

Autor: Mariano Galán Largo

A mis padres por apoyarme siempre.

A Mica, por ser mas que mi tutor.

A Susana, por estar junto a mí.

...Y en general, a todos que de una forma u otra han ayudado y colaborado en la realización del proyecto....

GRACIAS

Índice general

Capítulo 1

INTRODUCCIÓN

Una API (del inglés Application Programming Interface - Interfaz de Programación de Aplicaciones) es un conjunto de especificaciones de comunicación entre componentes software. Representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o iconos en la pantalla. De esta forma, los programadores se benefician de las ventajas de la API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio. Las APIs asimismo son abstractas: el software que proporciona una cierta API generalmente es llamado la implementación de esa API.

1.1. Presentación.

Muchas herramientas necesitan, en mayor o menor medida, la funcionalidad proporcionada por un editor de textos avanzado. Entre otras, las características necesarias son: el resaltado de sintaxis, resaltado de ciertas partes del documento (para mostrar resultado de búsquedas, errores, etc...), configuración del resaltado de los tokens lexicográficos, etc... Cada una de las herramientas requerirá una funcionalidad específica, pero en muchas ocasiones, existen muchas características comunes a todas ellas.

Por otro lado, todo editor de textos tiene una serie de funcionalidades que le permiten trabajar con ficheros. Por ejemplo, entre otras características es habitual poder abrir varios documentos a la vez, guardarlos todos, guardar como..., refrescar el contenido del disco duro, mostrar dos vistas sincronizadas del mismo documento, realizar búsquedas de cadenas de texto en varios documentos, disponer de una barra de herramientas configurable, etc... Por supuesto, cada tipo de herramienta necesitará de unas características propias, pero la mayoría de ellas serán comunes.

Este proyecto nace por la necesidad de construir herramientas de enseñanza de las tecnologías informáticas relacionadas con lenguajes de programación (Java, Pascal, C#, Hope, SML, etc...) y con los lenguajes de marcado (XML, HTML, XSL, CSS, DTD, XML-Schema, etc...).

1.2. Objetivos del proyecto.

El principal objetivo del proyecto es proporcionar, por un lado, un editor de textos al estilo de UltraEdit, jEdit o similar. Este editor de textos está diseñado de tal forma que es posible su modificación y/o ampliación de una forma clara y sencilla, puesto que el objetivo es adaptar su funcionalidad básica a cualquier herramienta educativa. También ha sido construido de forma modular, para que algunas partes puedan ser usadas en otras aplicaciones, por ejemplo, para incluir un editor de textos dentro de una presentación, en un libro electrónico o en un diagrama. Por último, este editor acepta ampliaciones mediante plugins. Los plugins permiten que se puedan incorporar nuevas funcionalidades a la misma aplicación desde distintas partes y distintos aspectos, y que cada una de estas funcionalidades puedan coexistir. El modelo de plugins es muy habitual en entornos de desarrollo y se está desarrollando un estándar llamado JSR-198 que define como debe hacerse un entorno que soporte plugins.

No se pretende la construcción de una herramienta desde cero, si no todo lo contrario. Antes del comienzo del desarrollo se deberá examinar exhaustivamente herramientas y/o APIs relacionadas para usarlas en el desarrollo del proyecto. Dichas herramientas han aportando rapidez y completitud a la herramienta final.

En este Proyecto de Fin de Carrera se define e implementa una API, llamada PlusProcessor, que representa y permite gestionar, entre otras cosas, la edición con resaltado de sintaxis de un lenguaje deseado mediante la integración de plugins de analizadores léxicos, los cuales ofrecerán soporte al sistema de resaltado. Está definida e implementada en lenguaje Java y sus objetivos principales son: Ofrecer una herramienta de edición de textos fácil de manipular y mantener, que permita una serie de funciones para resaltar sintaxis en un determinado entorno.

La API implementada permite abrir, modificar y guardar varios ficheros (existentes o nuevos) a la vez. Además se podrá asociar un resaltado de sintaxis a cada uno de los mismos. También tiene un sistema de plugins que permite incorporar nuevos lenguajes a la herramienta y esta diseñado de forma que la funcionalidad que se incorpore como plugin pueda ser además incorporada como parte de una herramienta independiente.

Para llevar a cabo estos requisitos, se ha realizado un estudio de herramientas de código abierto que proporcionen una funcionalidad parecida siempre y cuando se cumplan los requisitos de funcionalidad indicados, y sobre todo, los requisitos de modularidad y configurabilidad. Las herramientas a ser analizadas, en principio serán los proyectos de código abierto jEdit, NetBeans y Eclipse.

Capítulo 2

ESTADO DEL ARTE

2.1. Descripción de otras herramientas.

En la actualidad, hay disponibles multitud de editores de texto (comerciales, shareware opensource, etc) desarrollados en Java y en otros lenguajes. La funcionalidad existente en estos editores de texto se amplía en los entornos de desarrollo integrados de programación (IDE), las cuales incluyen características específicas del lenguaje del documento editado. Vamos a describir algunas de estas herramientas para así poder enumerar sus características más comunes:

UltraEdit (http://www.ultraedit.com/) es una herramienta con la que se pueden editar varios archivos a la vez de tamaño ilimitado y permite hacer cosas como el "reemplazar" de un texto en todo un conjunto de archivos. Soporta Java, ASPs, JSPs, HTML y hace el *sintax highlight* de palabras clave. Incluso tiene la posibilidad de entender el lenguaje Pl/SQL de Oracle. Dispone de macros y mucho más.

Emacs (http://www.gnu.org/software/emacs/) es un editor de texto altamente extensible y configurable distribuido bajo la licencia libre GPL. En la actualidad es mantenido por la Free Software Foundation. Forma parte del proyecto GNU.
Emacs es muy adecuado tanto para escribir texto plano como para programar o escribir scripts. Es extensible mediante el lenguaje elisp (Emacs Lisp), un dialecto Lisp.

jEdit (http://www.jedit.org/) es un editor de texto para programadores distribuido bajo los términos de la Licencia Pública General de GNU.

Dispone de docenas de plugins para diferentes áreas de aplicaciones. Soporta de forma nativa el coloreado de la sintaxis para más de 130 formatos de fichero. También se puede incluir nuevos formatos de forma manual utilizando ficheros XML.

Eclipse (http://www.eclipse.org/) es una poderosa herramienta que permite integrar diferentes aplicaciones para construir un entorno integrado de desarrollo (IDE). Es un proyecto de desarrollo de software open-source, que está dividido en tres partes: el proyecto Eclipse Proyect, Eclipse Tools, y Eclipse Technology Proyect.
Mediante Eclipse se pueden crear diversas aplicaciones como ser sitios Web, programas Java, C++ y Enterprise Java Beans. Su principal aplicación es JDT, herramienta para crear aplicaciones en Java.

Edit Pad (http://www.editpadpro.com/) es un editor de textos con soporte de múltiples ficheros, drag & drop, y más. Cambia entre los ficheros abiertos de forma fácil a través de pestañas. Este programa incorpora todas las funciones habituales en este tipo de programas: edición de ventanas múltiples, tamaño ilimitado de ficheros, hacer que la ventana esté encima de todas las demás, conversiones automáticas entre sistemas, etc.

XEditor (http://xeditor.uptodown.com/) es un sencillo procesador de textos que ocupa poco espacio y su interfaz es extremadamente simple. Entre sus características más notables destacan el resaltado de sintaxis para decenas de lenguajes de programación, reconocimiento automático (en base a la extensión de los archivos) de la sintaxis a utilizar, auto guardado de copias de seguridad de los documentos y auto completado de texto.

Gedit (http://www.gnome.org/projects/gedit/) es un completo editor de textos libre que se distribuye junto al gestor de escrito GNOME para sistemas tipo Unix. Este editor se caracteriza principalmente por su facilidad de uso, conseguida en gran parte gracias a un interfaz gráfico claro y limpio, mostrando únicamente las funcionalidades principales que suelen requerir la mayoría de usuarios. Además de las funcionalidades básicas que son habituales en un editor de texto, como copiar, cortar y pegar texto, imprimir ...etc, gedit incorpora, coloreado del texto según la sintaxis de varios lenguajes de programación,

incorporación de plugins para ampliar las funcionalidades básicas del programa, posibilidad de cambiar el color y fuente del texto del editor, búsqueda y reemplazo de texto y copia de seguridad de los ficheros sobre los que se trabaja.

Jext (http://www.freepoc.org/viewapp.php?id=8) es un programa que sirve para editar programas en Java y HTML, con completas funciones y herramientas de edición de código. Algunas de las funciones de este programa son: coloreado de sintaxis, función de guardar automática, tabla de herramientas, menú de inserción de sintaxis personalizable por el usuario, ilimitado número acciones a deshacer, permite realizar búsqueda de palabras y expresiones, etc.

Subpad (http://www.xtort.net/xtort/subpad.php) es un editor como el común Notepad de windows, con las mismas funciones sólo que incluye algunos añadidos. Incluye, por ejemplo, una utilidad de búsqueda y reemplazo de mayor eficacia, posibilidad de cerrarse con la tecla de ESC, reconocimiento de direcciones web, minimizado a la bandeja de sistema o dejando únicamente visible la barra de título, selección por columnas, etc.

SuperEdi (http://www.wolosoft.com/en/superedi/) es un editor de texto compuesto de múltiples ventanas de edición que, además, incluye un navegador de directorios, funciones que nos permiten seleccionar el color y el tipo de fuente de letra, un visor de HTML, así como una ayuda sobre este lenguaje de programación.

ConTEXT (http://www.context.cx/) es un sencillo editor de texto orientado especialmente como herramienta secundaria para programadores informáticos. El programa soporta los lenguajes C/C++, Delphi/Pascal, Java, Java Script, Visual Basic, Perl/CGI, HTML, SQL y FoxPro. ConTEXT no tiene límite de tamaño en los archivos que queramos crear ni en los que estén funcionando. Además, el programa soporta el uso de colores que diferencien los comandos de los lenguajes de programación.

2.2. Carencias encontradas.

Todos los procesadores de texto expuestos anteriormente tienen alguna característica común a los demás, habiendo en la mayoría de los casos, alguna otra que le diferencian del resto. Es por esta peculiaridad lo que hace que los usuarios de este tipo de herramientas se decanten por utilizar un procesador u otro, en función de sus necesidades.

En la gran mayoría de los casos, el procesador de textos es desarrollado con herramientas que facilitan el análisis lexicográfico de algunos lenguajes de programación, lo que limita su uso a determinados usuarios debido a esta exclusividad. Si avanzamos un poco más en el análisis de estas funcionalidades nos damos cuenta que actualmente existen pocas herramientas de procesamiento de textos que incorporen entre su diversidad de funcional (a través de nuevos módulos, plugins, etc) la posibilidad de integrar otros analizadores léxicos que, inclusive, puedan ser creados por el propio usuario de la herramienta.

Por otro lado, en la actualidad no existe una API que pueda ser utilizada como un componente más en la creación de un nuevo desarrollo software de cualquier nivel. Esta API esta especialmente diseñada para incorporar esta funcionalidad a otras herramientas de forma sencilla.

2.3. Conclusiones.

Desde el punto de vista expuesto en el punto 2.2, podríamos considerar interesante la posibilidad de crear una API que permita reunir las características más comunes existentes en todos los procesadores de textos, y que además, tenga la capacidad de incorporar nuevos *plugins* con nuevos motores de análisis lexicográfico, pudiendo ser desarrollados según exigencias del propio usuario.

También se podría suministrar al usuario la posibilidad de incorporar la API, en parte o en todo su conjunto, a otros desarrollos. Esto se consigue facilitando una interfaz de gestión que aporte los métodos necesarios para el manejo de toda la potencialidad del componente.

Capítulo 3

METODOLOGÍAS Y TECNOLOGÍAS

Gracias a una **metodología de trabajo** desarrollada para la elaboración de un producto, la capacidad de aprendizaje y **el uso de la tecnología** como herramienta, se ha diseñado una solución integral y modular que cumple con los objetivos y las expectativas de este proyecto.

3.1. Metodologías.

Cada proyecto es diferente al resto y por ello es necesario adaptar la metodología de desarrollo según su criticidad y complejidad.

PlusProcessor basa su metodología de desarrollo XP (eXtreme Programming), obteniendo un desarrollo más "pesado" o "liviano" según el proyecto lo necesite.

3.1.1. Extreme Programing.

La programación extrema (XP) es una de las metodologías de desarrollo de software más actuales en la cual se da por supuesto que es imposible prever todo antes de empezar a codificar. Es imposible capturar todos los requisitos del sistema, saber qué es todo lo que tiene que hacer ni hacer un diseño correcto al principio. Es bastante normal hacer un diseño, ponerse a codificar, ver que hay faltas o errores en el diseño, empezar a codificar fuera del diseño y al final el código y el diseño, o no se parecen, o hemos empleado mucho tiempo en cambiar la documentación de diseño para que se parezca al código.

En vez de tratar de luchar contra todo eso, lo asume y busca una forma de trabajar que se adapte fácilmente a esas circunstancias. Básicamente la idea de la programación extrema consiste en trabajar estrechamente con el cliente (en este

caso tutor), haciéndole mini-versiones con mucha frecuencia (cada dos semanas). En cada mini-versión se debe hacer el mínimo de código y lo más simple posible para que funcione correctamente. El diseño se hace sobre la marcha, haciendo un mini-diseño para la primera mini-versión y luego modificándolo en las siguientes mini-versiones. Además, no hay mejor documentación que el mismo código. El código, por tanto, también se modifica continuamente de mini-versión en mini-versión, añadiéndole funcionalidad y extrayendo sus partes comunes.

En un proyecto usando programación extrema se siguen más o menos los siguientes pasos:

El cliente junto al equipo de desarrollo definen qué es lo que se quiere hacer. Para ello utilizan las "historias de usuario". Una historia de usuario en un texto de una o dos frases en las que se dice algo que debe hacer el sistema. Es más extensa que un requisito (que suele ser una frase corta) y menos que un caso de uso (que puede ser de una o dos páginas). Se evalúa para cada historia de usuario el tiempo que puede llevar desarrollarla, que debe ser corto, de aproximadamente una semana. Un programador puede estimar con cierta fiabilidad un trabajo que le lleve unos días, pero la estimación es menos fiable si es de un plazo superior a una semana. Si es más largo, hay que partir la historia en otras más pequeñas. Luego se ordenan en el orden en que se van a desarrollar y se establecen las mini-versiones, de forma que cada mini-versión implementa varias de las historias de usuario.

Toda esta planificación va a ser, por supuesto, inexacta. No se puede saber todo lo que va a ser necesario ni evaluar los tiempos correctamente. La planificación deberá revisarse y modificarse continuamente a lo largo del proyecto. Las historias de usuario se modificarán, se quitarán o se añadirán nuevas sobre la marcha. Puesto que el cliente estará presente día a día durante todo el proyecto, verá el efecto y el esfuerzo necesario para las modificaciones pedidas y sabrá evaluar si merecen o no la pena.

Para las primeras historias que se van a implementar, se define una prueba para ver si la versión cumple perfectamente con la historia. Estas pruebas deben ser

automáticas, de forma que haya un programa de pruebas que ejecutemos y nos diga si la mini-versión es o no correcta.

Cada vez que se consigue codificar y que funcione una versión de usuario, se le da al cliente para que la vea, la pruebe y añada las posibles modificaciones para las siguientes mini-versiones. Cuando se realiza un mini-versión completa (compuesta por varias de las historias de usuario), incluso se entrega al usuario final para que empiece a trabajar con ella y reportar incidencias o mejoras.

Este ciclo se va repitiendo una y otra vez hasta que el cliente se de por satisfecho y cierre el proyecto. La planificación inicial puede hacerse escribiendo cada historia de usuario en una tarjeta, haciendo dos montones, las que ya están hechas y las que no. Se pueden tirar las tarjetas, añadir nuevas o cambiar las que ya hay. El número de tarjetas en cada montón nos da una idea exacta de cuánto hay hecho del proyecto.

Para que todo esto funcione, la programación extrema se basa en trece "prácticas básicas" que deben seguirse al pie de la letra (véase www.xprogramming.com/xpmag/whatisxp.htm).

Una información más completa sobre la Extreme Programing se encuentra en [7] y [8].

3.1.2. Proceso Unificado de Desarrollo (PUD).

El Proceso Unificado "*es un proceso de desarrollo de software configurable que se adapta a través de los proyectos variados en tamaños y complejidad. Se basa en muchos años de experiencia en el uso de la tecnología orientada a objetos en el desarrollo de software de misión crítica en una variedad de industrias por la compañía Rational*", donde confluyen 'los tres amigos' como se llaman a sí mismos o los tres grandes OO: Grady Booch, James Rumbaugh e Ivar Jacobson [M&R 1998].

El Proceso Unificado guía a los equipos de proyecto en cómo administrar el desarrollo iterativo de un modo controlado mientras se balancean los requerimientos del negocio, el tiempo de desarrollo y los riesgos del proyecto. El proceso describe los diversos pasos involucrados en la captura de los requisitos y en el establecimiento de una guía arquitectónica lo más pronto, para diseñar y probar el sistema hecho de acuerdo a los requisitos y a la arquitectura. El proceso describe qué entregables producir, cómo desarrollarlos y también provee patrones. El proceso unificado es soportado por herramientas que automatizan entre otras cosas, el modelado visual, la administración de cambios y las pruebas.

El Proceso Unificado ha adoptado un enfoque que se caracteriza por:

- Interacción con el usuario continua desde un inicio
- Mitigación de riesgos antes de que ocurran
- Liberaciones frecuentes
- Aseguramiento de la calidad
- Involucramiento del equipo en todas las decisiones del proyecto
- Anticiparse al cambio de requerimientos

El Proceso Unificado se enfoca en la arquitectura como el centro del desarrollo para asegurar que el desarrollo basado en componentes sea clave para un alto nivel de reuso, proporciona más detalle y guía para algunos de los roles en el proyecto.

Una vista arquitectónica es "*una descripción simplificada (una abstracción) de un sistema desde una perspectiva particular o punto de vista, que cubre particularidades y omite entidades que no son relevantes a esta perspectiva*" [Booch 1998].

Según el mismo autor, las características primordiales del Proceso Unificado son:

- Iterativo e incremental
- Centrado en la arquitectura
- Guiado por casos de uso
- Confrontación de riesgos

El Proceso Unificado es un proceso porque "*define quién está haciendo qué, cuándo lo hacer y cómo alcanzar cierto objetivo, en este caso el desarrollo de software*" [Jacobson 1998]. Según [Booch 1998], los conceptos clave del Proceso Unificado son:

Fase e iteraciones	¿Cuándo se hace?
Flujos de trabajo de procesos (actividades y pasos)	¿Qué se está haciendo?
Artefactos (modelos, reportes, documentos)	¿Qué se produjo?
Trabajador: un arquitecto	¿Quién lo hace?)

Las fases del ciclo de vida del software son: concepción, elaboración, construcción y transición. La concepción es definir el alcance del proyecto y definir el caso de uso. La elaboración es proyectar un plan, definir las características y cimentar la arquitectura. La construcción es crear el producto y la transición es transferir el producto a sus usuarios [Booch 1998].

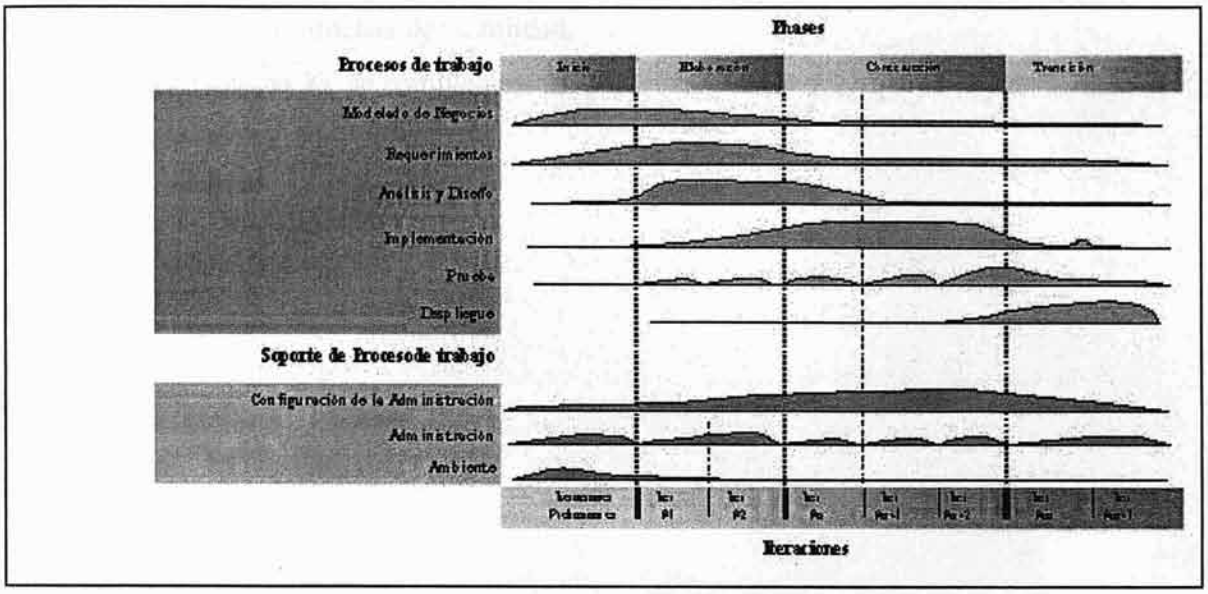


Figura 1. Estructura del Proceso Unificado

Una información más completa sobre el Proceso Unificado de Desarrollo se encuentra en [6].

3.1.3. Elección de la metodología.

Después de estudiar dos de las metodologías más usadas actualmente en el desarrollo software, se ha elegido el uso de XP. Los principales motivos de esta elección han sido:

- Equipo de desarrollo esta formado por una persona.
- El proyecto desarrolla una API que será utilizada por programadores, por tanto es más adecuado una metodología que integre al usuario final en el desarrollo.

- Un Proyecto Fin de Carrera tiene una duración estimada de 150 horas de trabajo (15 créditos). El tamaño de la aplicación construida es muy inferior a la gran mayoría de las aplicaciones comerciales.
- Dado que se trata de construir una API, es muy probable que se vayan incorporando nuevos requisitos a medida que se vayan necesitando. Por tanto habrá que volver a negociar los que se implementan.
- Es importante disponer de versiones funcionando a medida que se avanza en el desarrollo para realizar pruebas de viabilidad.

La metodología XP se complementará con UML para la descripción de los distintos diagramas.

3.2. Tecnologías.

3.2.1. Programación orientada a objetos.

Introducción al POO.

Actualmente una de las áreas más candentes en la industria y en el ámbito académico es la orientación a objetos. La orientación a objetos promete mejoras de amplio alcance en la forma de diseño, desarrollo y mantenimiento del software ofreciendo una solución a largo plazo a los problemas y preocupaciones que han existido desde el comienzo en el desarrollo de software: la falta de portabilidad del código y reusabilidad, código que es difícil de modificar, ciclos de desarrollo largos y técnicas de codificación no intuitivas.

Un lenguaje orientado a objetos ataca estos problemas. Tiene tres características básicas: debe estar basado en objetos, basado en clases y capaz de tener herencia de clases. Muchos lenguajes cumplen uno o dos de estos puntos; muchos menos cumplen los tres. La barrera más difícil de sortear es usualmente la herencia.

El concepto de programación orientada a objetos (POO) no es nuevo. Dado que la POO se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución del problema se realiza en términos de objetos, un lenguaje se dice que está basado en objetos si soporta objetos como una característica fundamental del mismo.

El elemento fundamental de la POO es, como su nombre lo indica, el objeto. Podemos definir un objeto como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.

Esta definición específica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bién estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

Un objeto puede considerarse como una especie de cápsula dividida en tres partes: Relaciones, propiedades y métodos.

Las relaciones permiten que el objeto se inserte en la organización y están formadas esencialmente por punteros a otros objetos.

Las propiedades distinguen un objeto determinado de los restantes que forman parte de la misma organización y tiene valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización.

Los métodos son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia.

Cada objeto es una estructura compleja en cuyo interior hay datos y programas, todos ellos relacionados entre sí, como si estuvieran encerrados conjuntamente en una cápsula. Esta propiedad (encapsulamiento), es una de las características fundamentales en la POO.

Los objetos son inaccesibles, e impiden que otros objetos, los usuarios, o incluso los programadores conozcan cómo está distribuida la información o qué información hay disponible. Esta propiedad de los objetos se denomina ocultación de la información.

Esto no quiere decir, sin embargo, que sea imposible conocer lo necesario respecto a un objeto y a lo que contiene. Si así fuera no se podría hacer gran cosa con él. Lo que sucede es que las peticiones de información a un objeto deben realizarse a través de mensajes dirigidos a él, con la orden de realizar la operación pertinente. La respuesta a estas órdenes será la información requerida, siempre que el objeto considere que quien envía el mensaje está autorizado para obtenerla.

El hecho de que cada objeto sea una cápsula facilita enormemente que un objeto determinado pueda ser transportado a otro punto de la organización, o

incluso a otra organización totalmente diferente que precise de él. Si el objeto ha sido bien construido, sus métodos seguirán funcionando en el nuevo entorno sin problemas. Esta cualidad hace que la POO sea muy apta para la reutilización de programas.

Una información más completa sobre la introducción a la programación orientada a objetos se encuentra en [5] y [11].

Organización de los objetos.

En principio, los objetos forman siempre una organización jerárquica, en el sentido de que ciertos objetos son superiores a otros de cierto modo.

Existen varios tipos de jerarquías: serán simples cuando su estructura pueda ser representada por medio de un "árbol". En otros casos puede ser más compleja.

- Polimorfismo

En programación orientada a objetos se denomina polimorfismo a la capacidad del código de un programa para ser utilizado con diferentes tipos de datos. También se puede aplicar a la propiedad que poseen algunas operaciones de tener un comportamiento diferente dependiendo del objeto (o tipo de dato) sobre el que se aplican.

El concepto de polimorfismo se puede aplicar tanto a funciones como a tipos de datos. Así nacen los conceptos de *funciones polimórficas* y *tipos polimórficos*. Las primeras son aquellas funciones que pueden evaluarse o ser aplicadas a diferentes tipos de datos de forma indistinta; los *tipos polimórficos*, por su parte, son aquellos tipos de datos que contienen al menos un elemento cuyo tipo no está especificado.

Se puede clasificar el polimorfismo en dos grandes clases:

-*Polimorfismo dinámico (o polimorfismo ad hoc)* es aquél en el que el código no incluye ningún tipo de especificación sobre el tipo de datos

sobre el que se trabaja. Así, puede ser utilizado a todo tipo de datos compatible.

-*Polimorfismo estático (o polimorfismo paramétrico)* es aquél en el que los tipos a los que se aplica el polimorfismo deben ser explicitados y declarados uno por uno antes de poder ser utilizados.

El polimorfismo dinámico unido a la herencia es lo que en ocasiones se conoce como programación genérica.

También se clasifica en herencia por redefinición de métodos abstractos y por método sobrecargado. El segundo hace referencia al mismo método con diferentes parámetros.

- Consideraciones finales

Día a día los costes del Hardware decrecen. Así surgen nuevas áreas de aplicación cotidianamente: procesamiento de imágenes y sonido, bases de datos multimedia, automatización de oficinas, ambientes de ingeniería de software, etc. Aún en las aplicaciones tradicionales encontramos que definir interfaces hombre-máquina suele ser bastante conveniente.

Lamentablemente, los costos de producción de software siguen aumentando; el mantenimiento y la modificación de sistemas complejos suele ser una tarea trabajosa; cada aplicación, (aunque tenga aspectos similares a otra) suele encararse como un proyecto nuevo, etc.

Todos estos problemas aún no han sido solucionados en forma completa. Pero como los objetos son portables (teóricamente) mientras que la herencia permite la reusabilidad del código orientado a objetos, es más sencillo modificar código existente porque los objetos no interaccionan excepto a través de mensajes; en consecuencia un cambio en la codificación de un objeto no afectará la operación con otro objeto siempre que los métodos respectivos permanezcan intactos. La introducción de tecnología de objetos como una herramienta conceptual para analizar, diseñar e implementar aplicaciones permite obtener aplicaciones más modificables, fácilmente extensibles y a partir de componentes

reusables. Esta reusabilidad del código disminuye el tiempo que se utiliza en el desarrollo y hace que el desarrollo del software sea mas intuitivo porque la gente piensa naturalmente en términos de objetos más que en términos de algoritmos de software .

Una información más completa sobre la organización de los objetos se encuentra en [12].

3.2.2. Java.

Introducción al Java.

Java es un lenguaje desarrollado por Sun con la intención de competir con Microsoft en el mercado de la red. Sin embargo, su historia se remonta a la creación de una filial de Sun (FirstPerson) enfocada al desarrollo de aplicaciones para electrodomésticos, microondas, lavaplatos, televisiones... Esta filial desapareció tras un par de éxitos de laboratorio y ningún desarrollo comercial.

Sin embargo, para el desarrollo en el laboratorio, uno de los trabajadores de FirstPerson, James Gosling, desarrolló un lenguaje derivado de C++ que intentaba eliminar las deficiencias del mismo. Llamó a ese lenguaje Oak. Cuando Sun abandonó el proyecto de FirstPerson, se encontró con este lenguaje y, tras varias modificaciones (entre ellas la del nombre), decidió lanzarlo al mercado en verano de 1995.

El éxito de Java reside en varias de sus características. Java es un lenguaje "sencillo", o todo lo sencillo que puede ser un lenguaje orientado a objetos, eliminando la mayor parte de los problemas de C++, que aportó su granito (o tonelada) de arena a los problemas de C. Es un lenguaje independiente de plataforma, por lo que un programa hecho en Java se ejecutará igual en un PC con Windows que en una estación de trabajo basada en Unix. También hay que destacar su seguridad, desarrollar programas que accedan ilegalmente a la memoria o realizar caballos de troya es una tarea propia de titanes.

Cabe mencionar también su capacidad multihilo, su robustez o lo integrado que tiene el protocolo TCP/IP, lo que lo hace un lenguaje ideal para

Internet. Pero es su sencillez, portabilidad y seguridad lo que le han hecho un lenguaje de tanta importancia.

-Portabilidad

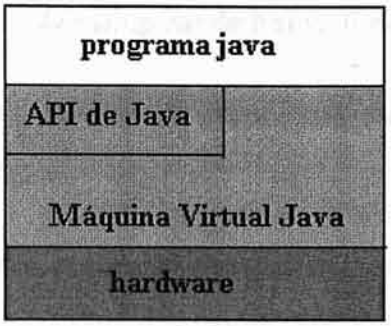
La portabilidad se consigue haciendo de Java un lenguaje medio interpretado medio compilado. ¿Cómo se come esto? Pues se coge el código fuente, se compila a un lenguaje intermedio cercano al lenguaje máquina pero independiente del ordenador y el sistema operativo en que se ejecuta (llamado en el mundo Java **bytecodes**) y, finalmente, se interpreta ese lenguaje intermedio por medio de un programa denominado máquina virtual de Java.

Este esquema lo han seguido otros lenguajes, como por ejemplo Visual Basic. Sin embargo, nunca se había empleado como punto de partida a un lenguaje multiplataforma ni se había hecho de manera tan eficiente. Cuando Java apareció en el mercado se hablaba de que era entre 10 y 30 veces más lento que C++. Ahora, con los compiladores JIT (**Just in Time**) se habla de tiempos entre 2 y 5 veces más lentos. Con la potencia de las máquinas actuales, esa lentitud es un precio que se puede pagar sin problemas contemplando las ventajas de un lenguaje portable.

-Orientación a objetos

Dado que Java es un lenguaje orientado a objetos, es imprescindible entender qué es esto y en qué afecta a nuestros programas. Desde el principio, la carrera por crear lenguajes de programación ha sido una carrera para intentar realizar abstracciones sobre la máquina. Al principio no eran grandes abstracciones y el concepto de lenguajes imperativos es prueba de ello. Exigen pensar en términos del ordenador y no en términos del problema a solucionar. Esto provoca que los programas sean difíciles de crear y mantener, al no tener una relación obvia con el problema que representan No abstraen lo suficiente [3].

Java es, por tanto, algo más que un lenguaje, ya que la palabra Java se refiere a dos cosas inseparables: el lenguaje que nos sirve para crear programas y la Máquina Virtual Java que sirve para ejecutarlos. Como vemos en la figura, el API de Java y la Máquina Virtual Java forman una capa intermedia (Java platform) que aísla el programa Java de las especificidades del hardware (hardware-based platform).



La máquina virtual de Java.

La Máquina Virtual Java (JVM) es el entorno en el que se ejecutan los programas Java, su misión principal es la de garantizar la portabilidad de las aplicaciones Java. Define esencialmente un ordenador abstracto y especifica las instrucciones (bytecodes) que este ordenador puede ejecutar. El intérprete Java específico ejecuta las instrucciones que se guardan en los archivos cuya extensión es .class. Las tareas principales de la JVM son las siguientes:

-Reservar espacio en memoria para los objetos creados.

-Liberar la memoria no usada (garbage collection).

-Asignar variables a registros y pilas.

-Llamar al sistema huésped para ciertas funciones, como los accesos a los dispositivos.

-Vigilar el cumplimiento de las normas de seguridad de las aplicaciones Java.

Esta última tarea, es una de las más importantes que realiza la JVM. Además, las propias especificaciones del lenguaje Java contribuyen extraordinariamente a este objetivo:

-Las referencias a arrays son verificadas en el momento de la ejecución del programa.

-No hay manera de manipular de forma directa los punteros

-La JVM gestiona automáticamente el uso de la memoria, de modo que no queden huecos.

-No se permiten realizar ciertas conversiones (casting) entre distintos tipos de datos.

Una información más completa sobre la máquina virtual de Java se encuentra en [4].

El Paquete Swing.

Los componentes del interfaz gráfico utilizan el nuevo modelo de Delegación de Eventos de Java. Todo ello redunda en una mayor funcionalidad en manos del programador, y en la posibilidad de mejorar en gran medida la presentación de los interfaces gráficos de usuario.

El paquete Swing es parte de la JFC (Java Foundation Classes) en la plataforma Java. La JFC provee facilidades para ayudar a la gente a construir GUIs. Swing abarca componentes como botones, tablas, marcos, etc...

Swing aprovecha la circunstancia de que sus componentes no tienen sus propiedades asignadas ni calculadas antes de ser mostrados por la pantalla para soportar lo que llaman "*pluggable look and feel*", es decir, que la apariencia de la aplicación se adapte dinámicamente al sistema operativo y plataforma en que esté corriendo .

Una información más completa sobre el paquete Swing de Java se encuentra en [10].

3.2.3. JFlex.

Introducción al análisis léxico.

El análisis léxico consiste en identificar en un texto aquellas cadenas que se ajustan (encajan en) a un determinado patrón lingüístico.

Las expresiones regulares son uno de los medios más habituales para representar patrones léxicos y representa un conjunto de cadenas mediante una expresión en la que se mezclan símbolos y operadores.

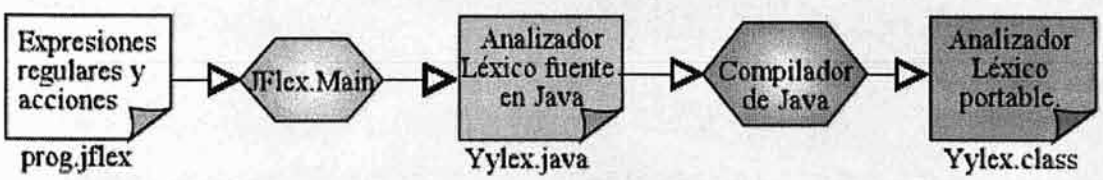
La herramienta JFlex.

JFlex es un generador de analizadores lexicográficos desarrollado por Gerwin Klein como extensión a la herramienta JLex desarrollada en la Universidad de Princeton. JFlex está desarrollado en Java y genera código Java.

Los programas escritos para JFlex tienen un formato parecido a los escritos en Lex, los cuales son también admitidos por JFlex.

La instalación y ejecución de JFlex es trivial. Una vez descomprimido el Fichero **jflex-1.3.5.zip**, dispondremos del fichero JFlex.jar que tan sólo es necesario en tiempo de meta-compilación, siendo el analizador generado totalmente independiente.

La clase Main del paquete JFlex es la que se encarga de metacompilar nuestro programa .jflex de entrada; de esta manera, una invocación típica es de la forma: java JFlex.Main fichero.jflex lo que generará un fichero Yylex.java que implementa al analizador lexicográfico. En la siguiente figura se ilustra el proceso a seguir, y cómo el nombre por defecto de la clase generada es **Yylex**:



Obtención de un programa portable en Java a partir de una especificación JFlex

Areas en JFlex.

Las áreas permiten indicar a JFlex una serie de opciones para adaptar el fichero **.java** resultante de la meta-compilación y que será el que implemente nuestro analizador lexicográfico en Java. También permite asociar un identificador de usuario a los patrones más utilizados en el área de reglas.

- **Opciones**

  Las opciones más interesantes se pueden clasificar en opciones de clase, de la función de análisis, de fin de fichero, de juego de caracteres y de contadores. Todas ellas empiezan por el carácter **%** y no pueden estar precedidas por nada en la línea en que aparecen.

- **Declaraciones.**

  Además de opciones, el programador puede indicar declaraciones de dos tipos en el área que nos ocupa, a saber, declaraciones de estados léxicos y declaraciones de reglas.

- **Reglas.**

  El área de reglas tiene la misma estructura que en Lex, con la única diferencia de que es posible agrupar las reglas a aplicar en un mismo estado léxico.

Funciones y variables de la clase Yylex.

Se tratan de funciones y variables miembro, por lo que, si son utilizadas fuera de las acciones léxicas deberá indicarse el objeto contextual al que se aplican (p.ej. **miAnalizador.yylex()**).

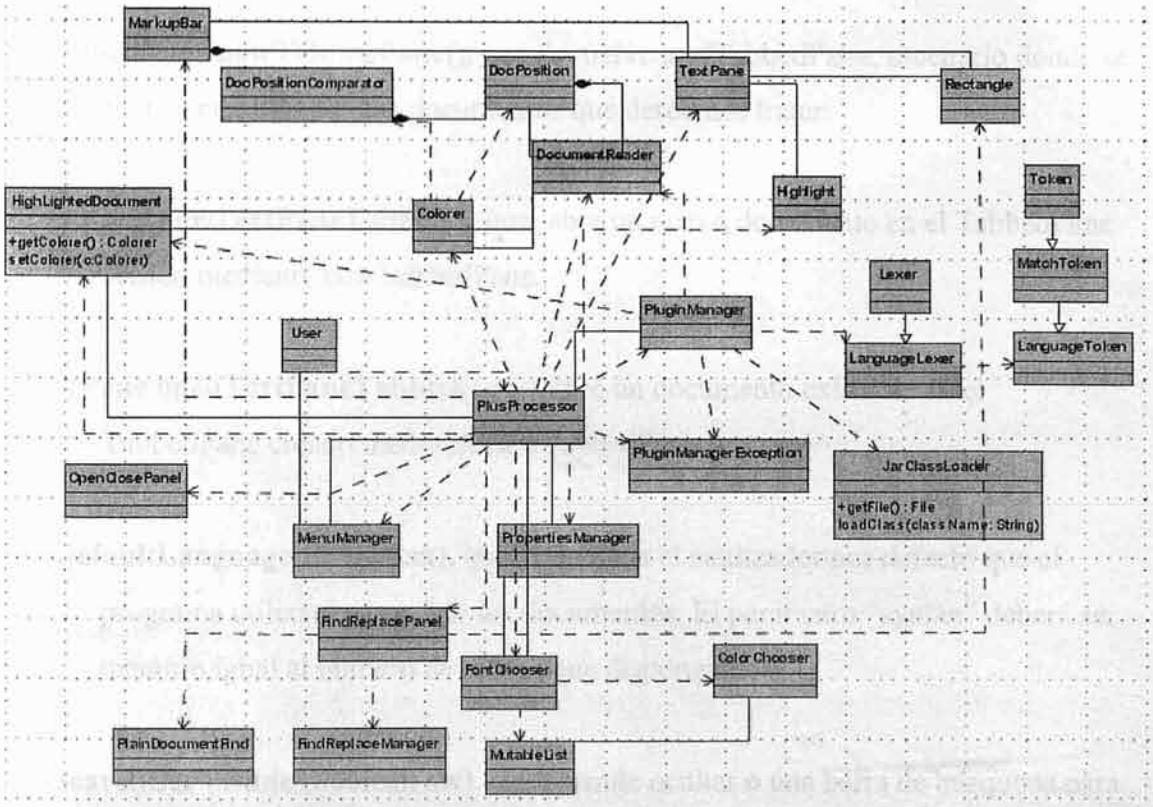
Capítulo 4

LA HERRAMIENTA PLUS PROCESSOR

En este capítulo vamos a describir cuales son los mecanismos aportados por la herramienta para obtener un uso provechoso de la misma.

Existen dos tipos de usuarios, los de aplicación del procesador de textos y los desarrolladores que usan la API. Nos centraremos en el desglosamiento de las clases principales que implican una interactuación directa con el segundo tipo de usuarios (desarrolladores que usan la API), por ser el objetivo principal de este proyecto. Como veremos en posteriores apartados, la API suministra un componente de procesamiento de textos totalmente configurable, y pone a disposición de los usuarios los métodos apropiados para su gestión de forma controlada.

4.1. Diagramas de clases.



4.2. Clase principal PlusProcessor.

Mediante esta clase podemos vincular todos los documentos que deseemos abrir o crear a la potencialidad de la herramienta. Después de realizar una nueva instancia de esta clase, podremos realizar las siguientes funcionalidades:

-getMenuManager(), que devuelve el menú estándar para la manipulación de los documentos que se deseen manejar.

-TextPane newTextPane(), que abre un nuevo documento de texto y devuelve el objeto que lo contiene.

-TextPane openTextPane(), que abre un documento de texto existente y devuelve el objeto que lo contiene.

-public Vector getSinglePane (), que devuelve un vector con todos los TextPane que abrimos con el modo newTextPane/openTextPane además de los abiertos mediante el menú.

-JTabbedPane newTabbedPane(), que devuelve un JTabbedPane, escenario donde se representarán todos los documentos que deseemos tratar.

-TextPane newTextPaneTabbed(), que abre un nuevo documento en el TabbedPane creado mediante newTabbedPane.

-TextPane openTextPaneTabbed(), que abre un documento existente en el TabbedPane creado mediante newTabbedPane.

-setDefaultLanguage(int syntax), que selecciona el analizador por defecto que el programa utilizará para abrir los documentos. El parámetro "syntax" deberá ser menor o igual al número de plugin que dispongamos.

-setSearchBarVisible (boolean sw), que permite ocultar o una barra de búsqueda para una mejor visualización global de las cadenas buscadas.

-public TextPane getTextArea (), que devuelve el área de texto seleccionada en el JTabbedPane.

-public Vector underlineWord(String word, int cont, TextPane areaTexto), función que subraya las primeras "cont" palabras que coincidan con la cadena "word" en un determinado TextPane y devuelve su vector de coordenadas.

-public void noUnderlineWord(Vector vector,TextPane areaTexto), que elimina las palabras subrayadas correspondientes al vector de coordenadas obtenido mediante la función underlineWord y que es asociado a un determinado componente de texto TextPane.

-public Object underlineIndex (int idx0, int idx1,TextPane areaTexto), que subraya todo el texto contenido desde la posición idx0 e idx1 en un TextPane y devuelve un objeto con sus coordenadas.

-public void noUnderlineIndex(Object obj,TextPane areaTexto), que elimina las palabras subrayadas correspondientes al objeto de coordenadas obtenido mediante la función underlineIndex y que es asociado a un determinado componente de texto TextPane.

Notas a tener en cuenta:

Se deberá tener muy en cuenta que una sola clase PlusProcessor solamente debe encargarse de manejar o bien paneles de texto sueltos o bien un JTabbedPane. De lo contrario, la herramienta pierde su funcionalidad y comenzará a realizar un mal funcionamiento.

Los paneles de texto sueltos no llevan incorporada la funcionalidad de la SearchBar.

La funcionalidad New y Open en el menú cuando utilizamos TextPane sueltos no tendrán ninguna repercusión a efectos prácticos. Sería conveniente ocultar estas opciones cuando se manejan TextPane de esta forma.

Es importante un cierre ordenado de la clase PlusProcessor. Para ello es buena práctica invocar el método "miCloseAll.miCloseAll()" en la instancia principal. A continuación detallamos un ejemplo donde realizamos esta acción al detectar un cierre de la aplicación en nuestro programa principal:

```
        …
        PlusProcessor plusProcessor;
        …
        class ActionWindow extends WindowAdapter {
                    public void windowClosing( WindowEvent evt ) {
                          // ¡¡¡Importante!!!
                          plusProcessor.miCloseAll();
                          System.exit(0);
                    }
}
```

4.3. Clase MenuManager.

Esta clase se encarga de gestionar y manejar las opciones de un menú vinculante a todas las funcionalidades de la herramienta. Después de obtener este menú mediante el método getMenuManager de la librería principal (PlusProcessor), podemos configurar el menú general mediante los siguientes métodos:

-setNewVisible(boolean visible), que permite ocultar o mostrar la opción New (abrir documentos nuevos) del submenú File.

-setOpenVisible(boolean visible), que permite ocultar o mostrar la opción Open (abrir documentos existentes) del submenú File.

-setSaveVisible(boolean visible), que permite ocultar o mostrar la opción Save (guardar documentos abiertos) del submenú File.

-setSaveAsVisible(boolean visible), que permite ocultar o mostrar la opción Salvar como (guardar documentos abiertos con un nombre) del submenú File.

-setSaveAllVisible(boolean visible), que permite ocultar o mostrar la opción Save All (guardar todos los documentos abiertos) del submenú File.

-setCloseVisible(boolean visible), que permite ocultar o mostrar la opción Close (cerrar documentos abiertos) del submenú File.

-setCloseAllVisible(boolean visible), que permite ocultar o mostrar la opción Close All (cerrar todos los documentos abiertos) del submenú File.

-setExitVisible(boolean visible), que permite ocultar o mostrar la opción Exit (salir de la aplicación) del submenú File.

-setUndoRedoVisible(boolean visible), que permite ocultar o mostrar las opciones Undo y Redo (deshacer y rehacer cambios) del submenú Edit.

-setCutCopyPasteSelectVisible(boolean visible), que permite ocultar o mostrar las opciones Cut, Copy y Paste (cortar, copiar y pegar texto) del submenú Edit.

-setFindVisible(boolean visible), que permite ocultar o mostrar las opciones Find/Replace, Find Next y Find All (buscar/remplazar, buscar siguiente y buscar todas las cadenas que coincidan con una expresión) del submenú Edit.

-setFontVisible(boolean visible), que permite ocultar o mostrar las opciones Font Size Up y Font Size Down (aumentar o disminuir el tamaño de las letras del texto) del submenú Edit.

-JList getJMenues(), que devuelve una lista con todos los submenus disponibles para que sean manejados por el usuario como mejor le convenga.

4.4. Ejemplos de utilización de la herramienta.

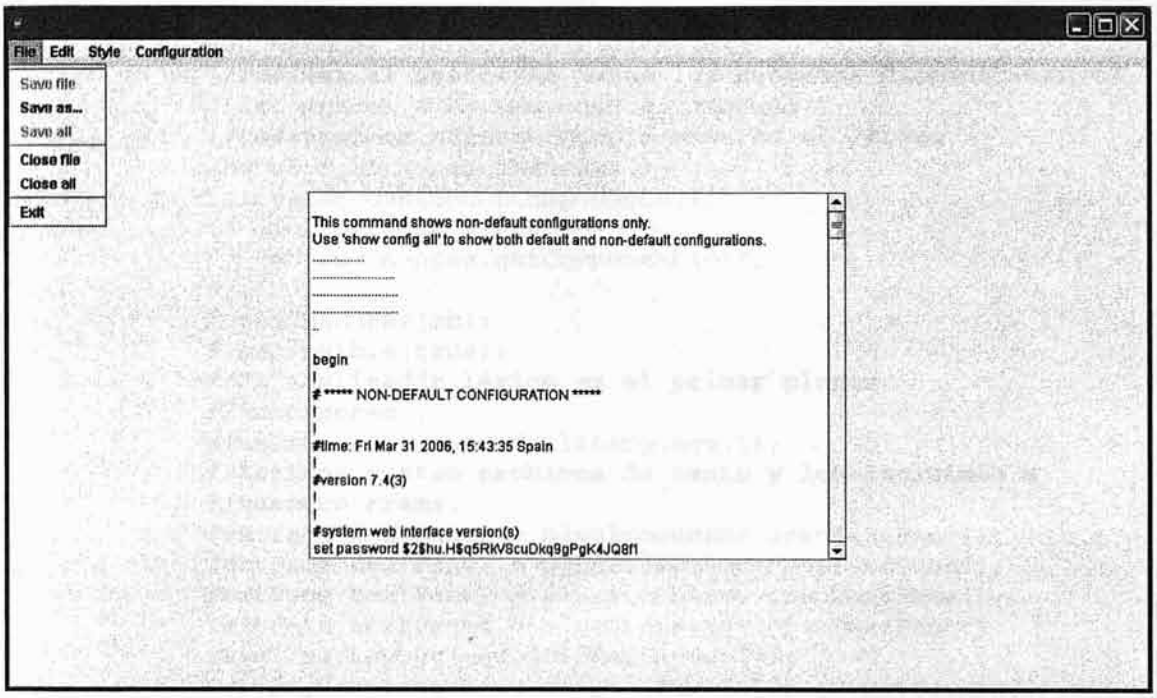
- **Ejemplo 1.**

Vamos a realizar una aplicación creando un contenedor `JFrame`, donde agregaremos un documento TextPane (el cual abriremos con la clase PlusProcessor) y un menú para realizar funciones sobre el documento (ver figura 1). Describiremos las acciones más importantes del ejemplo.

```
import java.awt.*;
import java.awt.GridBagConstraints;
import javax.swing.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class User extends JPanel
{
      PlusProcessor plusProcessor;
      public User()
      {
            //Creamos una instancia de PlusProcessor.
            plusProcessor = new PlusProcessor();
            JFrame f = new JFrame();
            f.addWindowListener(new ActionWindow());
            JPanel panel = new JPanel(new BorderLayout());
            //Pedimos a la instancia de PlusProcessor su gestor
            //de menús.
            MenuManager menuManager = plusProcessor.getMenuManager();
            //Indicamos al gestor de menus que la opción New no sea
            //visible.
            menuManager.setNewVisible(false);
            //Indicamos al gestor de menus que la opción Open no sea
      //visible.
            menuManager.setOpenVisible(false);
            //Pedimos al gestor de menus los submenus disponibles.
            JList menues = menuManager.getJMenues();
            //Construimos nuestro propio menu en el JFrame.
            JMenuBar mb = new JMenuBar();
            int j= menues.countComponents();
            for(int i=0; i< j; i++){
                mb.add(menues.getComponent(0));
            }
            f.setJMenuBar(mb);
            f.setVisible(true);
            //El analizador léxico será el 0 (por defecto).
            plusProcessor.setDefaultLanguage(0);
            //Abrimos un archivo de texto y lo incluimos a nuestro
            //Frame.
            TextPane textPane1 = plusProcessor.openTextPane();
            panel.setLayout(new GridBagLayout());
            GridBagConstraints config1 = new GridBagConstraints();
            config1.ipadx= 450;
            config1.ipady= 300;
            config1.gridx = 0;
```

```
            config1.gridy = 0;
            panel.add(new JScrollPane(textPane1), config1);
            f.setSize(1000, 600);
            f.setContentPane(panel);
        }
        class ActionWindow extends WindowAdapter {
            public void windowClosing( WindowEvent evt ) {
                  // ¡¡¡Importante!!!
                  plusProcessor.miCloseAll();
                  System.exit(0);
            }
         }
          public static void main(String args[]){
               new User();
         }
}
```

Figura 1



- **Ejemplo 2.**

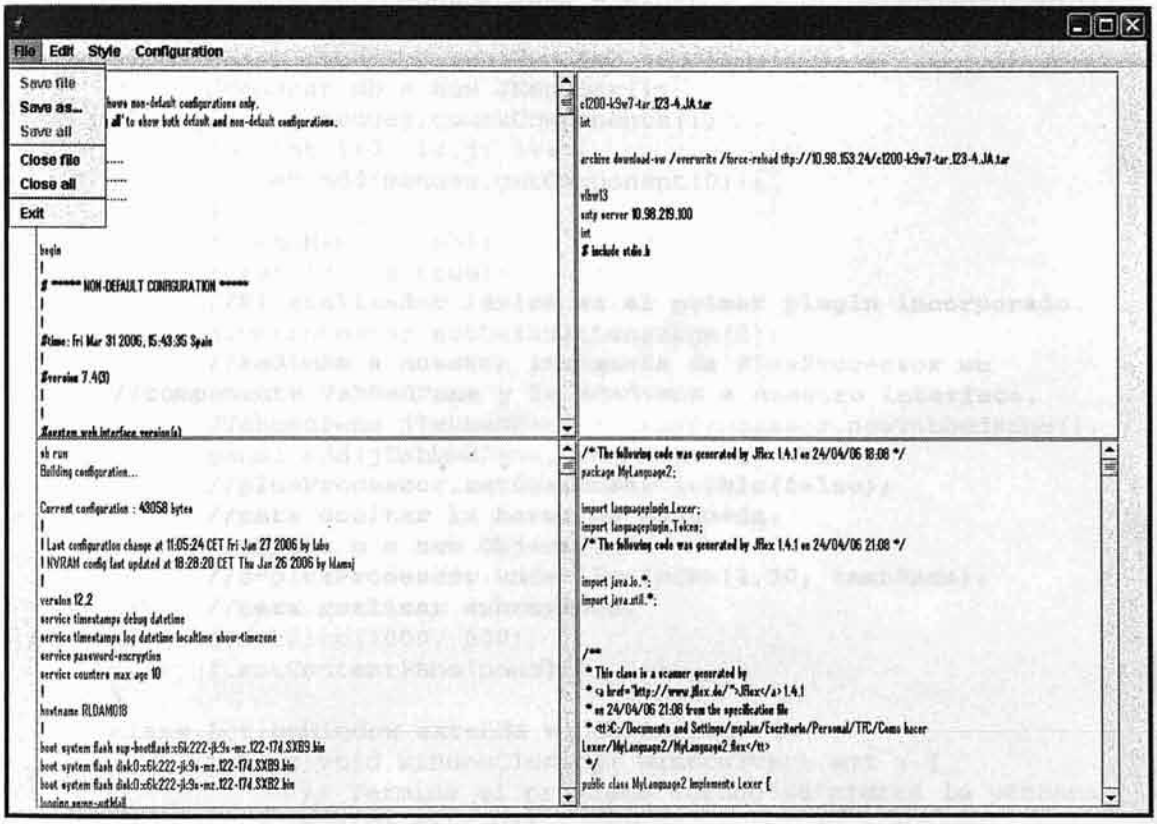
En el siguiente ejemplo, crearemos una aplicación creando un contenedor JFrame, donde agregaremos cuatro documentos TextPane y un menú para realizar funciones sobre los documentos (ver figura 2). Describiremos las acciones más importantes del ejemplo.

```
import java.awt.*;
import java.awt.GridBagConstraints;
import javax.swing.*;
```

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class User extends JPanel
{
      PlusProcessor plusProcessor;
      public User()
      {
            //Creamos una instancia de PlusProcessor.
            plusProcessor = new PlusProcessor();
            JFrame f = new JFrame();
            f.addWindowListener(new ActionWindow());
            JPanel panel = new JPanel(new BorderLayout());
            //Pedimos a la instancia de PlusProcessor su gestor de
            //menús.
            MenuManager menuManager = plusProcessor.getMenuManager();
      //Indicamos al gestor de menus que la opción New no sea
            //visible.
            menuManager.setNewVisible(false);
            //Indicamos al gestor de menus que la opción Open no sea
            //visible.
            menuManager.setOpenVisible(false);
            //Pedimos al gestor de menus los submenus disponibles.
            JList menues = menuManager.getJMenues();
            //Construimos nuestro propio menu en el JFrame.
            JMenuBar mb = new JMenuBar();
            int j= menues.countComponents();
            for(int i=0; i< j; i++){
                mb.add(menues.getComponent(0));
            }
            f.setJMenuBar(mb);
            f.setVisible(true);
            //El analizador léxico es el primer plugin
            //incorporado.
            plusProcessor.setDefaultLanguage(1);
            //Abrimos cuatro archivos de texto y los incluimos a
            //nuestro Frame.
            TextPane textPane1 = plusProcessor.openTextPane();
            TextPane textPane2 = plusProcessor.openTextPane();
            TextPane textPane3 = plusProcessor.openTextPane();
            TextPane textPane4 = plusProcessor.openTextPane();
            panel.setLayout(new GridBagLayout());
            GridBagConstraints config1= new GridBagConstraints();
            config1.ipadx= 450;
            config1.ipady= 300;
            config1.gridx = 0;
            config1.gridy = 0;
            panel.add(new JScrollPane(textPane1), config1);
            GridBagConstraints config2= new GridBagConstraints();
            config2.ipadx= 450;
            config2.ipady= 300;
            config2.gridx = 1;
            config2.gridy = 0;
            config2.fill = GridBagConstraints.BOTH;
            panel.add(new JScrollPane(textPane2), config2);
            GridBagConstraints config3= new GridBagConstraints();
            config3.ipadx= 450;
            config3.ipady= 300;
            config3.gridx = 0;
            config3.gridy = 1;
            config3.fill = GridBagConstraints.BOTH;
            panel.add(new JScrollPane(textPane3), config3);
```

```
            GridBagConstraints config4= new GridBagConstraints();
            config4.ipadx= 450;
            config4.ipady= 300;
            config4.gridx = 1;
            config4.gridy = 1;
            config4.fill = GridBagConstraints.BOTH;
            panel.add(new JScrollPane(textPane4), config4);
            f.setSize(1000, 600);
            f.setContentPane(panel);
      }
      class ActionWindow extends WindowAdapter {
            public void windowClosing( WindowEvent evt ) {
                  // ¡¡¡Importante!!!
                  plusProcessor.miCloseAll.doClick();
                  System.exit(0);
            }
      }
      public static void main(String args[])
      {
              new User();
       }
}
```

Figura 2



- **Ejemplo 3.**

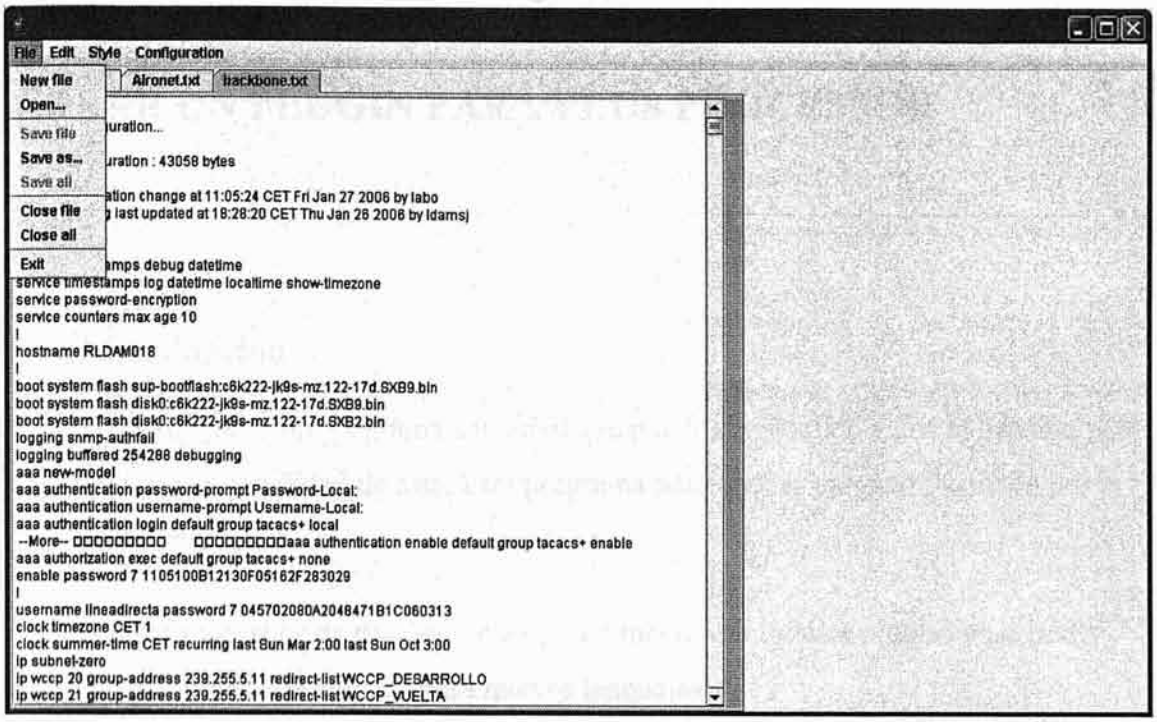
Por último, vamos a realizar una aplicación creando un contenedor JFrame,

donde agregaremos un componente JTabbedPane y un menú para realizar funciones sobre los documentos contenidos en los paneles (ver figura 3). Describiremos las acciones más importantes del ejemplo.

```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.*;

public class User extends JPanel
{
      PlusProcessor plusProcessor;
      public User()
      {
            //Creamos una instancia de PlusProcessor
            plusProcessor = new PlusProcessor();
            JFrame f = new JFrame();
            f.addWindowListener(new ActionWindow());
            JPanel panel = new JPanel(new BorderLayout());
            //Pedimos a la instancia de PlusProcessor su gestor de
      //menús.
            MenuManager menuManager = plusProcessor.getMenuManager();
            //Pedimos al gestor de menus los submenus disponibles.
            JList menues = menuManager.getJMenues();
            JMenuBar mb = new JMenuBar();
            int j= menues.countComponents();
            for(int i=0; i< j; i++){
                mb.add(menues.getComponent(0));
            }
            f.setJMenuBar(mb);
            f.setVisible(true);
            //El analizador léxico es el primer plugin incorporado.
            plusProcessor.setDefaultLanguage(2);
            //Pedimos a nuestra instancia de PlusProcessor un
      //componente TabbedPane y la añadimos a nuestro interface.
            JTabbedPane jTabbedPane = plusProcessor.newTabbedPane();
            panel.add(jTabbedPane, BorderLayout.WEST);
            //plusProcessor.setSearchBarVisible(false);
            //para ocultar la barra de busqueda.
            //Object o = new Object();
            //o=plusProcessor.underlineIndex(1,10, textPane);
            //para realizar subrayados.
            f.setSize(1000, 600);
            f.setContentPane(panel);
      }
      class ActionWindow extends WindowAdapter {
            public void windowClosing( WindowEvent evt ) {
                  // Termina el programa cuando se cierra la ventana
                  System.exit(0);
            }
      }
      public static void main(String args[])
      {
            new User();
      }
}
```

Figura 3



Los documentos de texto podemos abrirlos mediante el menú de la aplicación creada.

Capítulo 5

CREAR UN PLUGIN PARA PLUS PROCESSOR

5.1. Introducción.

Un *plugin* es un programa adicional que puede ser añadido a una aplicación para aumentar la funcionalidad de esta. Este programa adicional es cargado y corrido por la aplicación principal.

PlusProcessor esta diseñado para poder incorporar nuevos plugins y así poder integrar coloreados de sintaxis para nuevos lenguajes.

5.2. Diseño de un lenguaje compatible mediante JFlex.

Vamos a diseñar un pequeño analizador léxico que identifique las palabras reservadas "char", "for", "int", "if", "public", "switch" y "while", los operadores "=", "+", "-", "*" y "/", los decimales enteros y todos los identificadores. Para ello, debemos de entender como están declaradas las diversas clases aportadas por la herramienta, así como sus relaciones de herencia, para una buena integración de un analizador.
El analizador léxico de un lenguaje en concreto consta de las siguientes partes:

5.2.1. Clases de PlusProcessor para el desarrollo de plugins.

La clase Abstracta **Token**, es la clase padre de todos los Token de un analizador léxico. Los métodos abstractos son utilizados por la herramienta y deben siempre ser especificados. Veamos cuales son sus funcionalidades:

- **int getID()**, el cual devuelve el identificador del token.
- **string getDescription()**, el cual devuelve la descripción de este token.
- **string getContents()**, que devuelve la cadena de texto que representa el token.

- **boolean isComment()**, que determinará si el token es un comentario, retornando true en caso de serlo.
- **boolean isWhiteSpace()**, que retornará trae si el token representa un espacio en blanco.
- **boolean isError()**, el cual retorna trae si el token es un error.
- **int getLineNumber()**, que retorna la posición de la entrada en la cual se encontró el token.
- **int getCharBegin()**, retorna el desplazamiento de caracteres dentro de la entrada en la cual el token comenzó.
- **int getCharEnd()**, retorna el desplazamiento de caracteres dentro de la entrada en la cual el token finalizó.
- **string errorString()**, el estado de el token es indefinido y devuelve la cadena encontrada.
- **int getState()**, retorna un entero que representa el estado del token.

Presenta los siguientes atributos:

- **int UNDEFINED_STATE = -1**, el estado del token es indefinido.
- **int INITIAL_STATE = 0**, el cual es el inicial estado del token.

La clase Abstracta **Lexer**, la cual será la base de referencia para realizar cualquier analizador léxico con jflex. Se definen los métodos siguientes:

- **token getNextToken()**, el cual retorna el siguiente token encontrado en la cadena de entrada)
- **reset (java.io.Reader reader, , int yyline, int yychar, int yycolumn)** el cual resetea todos los valores del analizador tales como la cadena de entrada, el número de línea, la posición dentro de la línea, y la columna del primer token.
- **public Vector getTokens()**, que retorna un vector con el nombre de cada uno de los posibles tokens que es capaz devolver el analizador.

En un segundo nivel de herencia, la herramienta aporta la clase Abstracta **MatchToken**, que puede ser utilizada por el usuario como apoyo en la implementación

de sus clases de tokens, si es que la existente no satisface sus necesidades. Esta clase hereda de Token, y en ella se codifican todos los métodos de la clase padre. También se han definido los atributos que mantendrá la propia clase para guardar los datos del token que serán consultados a través de los métodos de la misma:

private int ID, para el número del token representado.

private String contents, que representara el texto de el token.

private int lineNumber, que será el número de línea donde comienza el token.

private int charBegin, desplazamiento dentro de la línea donde comienza el token.

private int charEnd, desplazamiento dentro de la línea donde finaliza el token.

private int state, estado del token.

5.2.2. Implementación de un analizador para un lenguaje concreto.

Vamos a definir su estructura mediante JFlex, que dará forma a una clase que heredará de la clase abstracta Lexer. También se deberá predefinir los token mediante otra clase que heredará de la clase abstracta MatchToken.

La clase hija **LanguageSymbol** es un token que es retornado por una clase lexer que esta analizando un fichero fuente escrito en nuestro lenguaje "MyLanguage":

```
1.    package MyLanguage;
2.    import java.util.Vector;
3.    import languageplugin.MatchToken;
4.    public class LanguageSymbol extends MatchToken{
5.    public final static int FOR = 0x101;
6.    public final static int INT = 0x102;
7.    public final static int IF = 0x103;
8.    public final static int PUBLIC = 0x104;
9.    public final static int SWITCH = 0x105;
10.   public final static int WHILE = 0x106;
11.   public final static int IDENTIFIER = 0x201;
12.   public final static int INTEGER_LITERAL = 0x0301;
13.   public final static int CHAR = 0x302;
14.   public final static int PLUS = 0x500;
15.   public final static int MINUS = 0x501;
16.   public final static int MULT = 0x502;
17.   public final static int DIV = 0x503;
18.   public final static int EQ = 0x504;
19.   public final static int WHITESPACE = 0xE00;
20.   public final static int UNKNOWN = 0xF00;
21.   //public final static int EOF = 0xFF00;
22.   /**
```

```
23.    create a default constructor.
24.    */
25.    public LanguageSymbol(){}
26.    /**
27.    Create a new token.
28.    The constructor is typically called by the lexer
29.    *
30.    @param ID the id number of the token
31.    @param contents A string representing the text of the token
32.    @param lineNumber the line number of the input on which this
       token started
33.    @param charBegin the offset into the input in characters at
       which this token started
34.    @param charEnd the offset into the input in characters at which
       this token ended
35.    */
36.    public LanguageSymbol(int ID, String contents, int lineNumber,
       int charBegin, int charEnd){
37.          this (ID, contents, lineNumber, charBegin, charEnd,
       MatchToken.UNDEFINED_STATE);
38.    }

39.    /**
40.    Create a new token.
41.    The constructor is typically called by the lexer
42.    *
43.    @param ID the id number of the token
44.    @param contents A string representing the text of the token
45.    @param lineNumber the line number of the input on which this
       token started
46.    @param charBegin the offset into the input in characters at
       which this token started
47.    @param charEnd the offset into the input in characters at which
       this token ended
48.    @param state the state the tokenizer is in after returning this
       token.
49.    */
50.    public LanguageSymbol(int ID, String contents, int lineNumber,
       int charBegin, int charEnd, int state){
51.          super (ID, contents, lineNumber, charBegin, charEnd,
       state);
52.    }

53.    /**
54.    Checks this token to see if it is a reserved word.
55.    Reserved words are explained in <A H
       ref=http://java.sun.com/docs/books/jls/html/>Java
56.    Language Specification</A>.
57.    *
58.    @return true if this token is a reserved word, false otherwise
59.    */
60.    public boolean isReservedWord(){
61.    return((ID >> 8) == 0x1);
62.    }
63.    /**
64.    Checks this token to see if it is an identifier.
65.    Identifiers are explained in <A
       Href=http://java.sun.com/docs/books/jls/html/>Java
66.    Language Specification</A>.
67.    *
68.    @return true if this token is an identifier, false otherwise
```

```
69.    */
70.    public boolean isIdentifier(){
71.         return((ID >> 8) == 0x2);
72.    }

73.    /**
74.    Checks this token to see if it is a literal.
75.    Literals          are          explained          in          <A
       Href=http://java.sun.com/docs/books/jls/html/>Java
76.    Language Specification</A>.
77.    *
78.    @return true if this token is a literal, false otherwise
79.    */
80.    public boolean isLiteral(){
81.         return((ID >> 8) == 0x3);
82.    }
83.    /**
84.    Checks this token to see if it is a Separator.
85.    Separators          are          explained          in          <A
       Href=http://java.sun.com/docs/books/jls/html/>Java
86.    Language Specification</A>.
87.    *
88.    @return true if this token is a Separator, false otherwise
89.    */
90.    public boolean isSeparator(){
91.         return((ID >> 8) == 0x4);
92.    }
93.    /**
94.    Checks this token to see if it is a Operator.
95.    Operators          are          explained          in          <A
       Href=http://java.sun.com/docs/books/jls/html/>Java
96.    Language Specification</A>.
97.    *
98.    @return true if this token is a Operator, false otherwise
99.    */
100.   public boolean isOperator(){
101.        return((ID >> 8) == 0x5);
102.   }
103.   /**
104.   Checks this token to see if it is a comment.
105.   *
106.   @return true if this token is a comment, false otherwise
107.   */
108.   public boolean isComment(){
109.        return((ID >> 8) == 0xD);
110.   }
111.   /**
112.   Checks this token to see if it is White Space.
113.   Usually tabs, line breaks, form feed, spaces, etc.
114.   *
115.   @return true if this token is White Space, false otherwise
116.   */
117.   public boolean isWhiteSpace(){
118.        return((ID >> 8) == 0xE);
119.   }
120.   /**
121.   Checks this token to see if it is an Error.
122.   Unfinished comments, numbers that are too big, unclosed strings,
       etc.
123.   *
124.   @return true if this token is an Error, false otherwise
```

```
125.   */
126.   public boolean isError(){
127.        return((ID >> 8) == 0xF);
128.   }
129.   /**
130.   A description of this token.  The description should
131.   be appropriate for syntax highlighting.  For example
132.   "comment" is returned for a comment.
133.   *
134.   @return a description of this token.
135.   */
136.   public String getDescription(){
137.        if (isReservedWord()){
138.        return("reservedWord");
139.        } else if (isIdentifier()){
140.             return("identifier");
141.             } else if (isLiteral()){
142.                  return("literal");
143.                  } else if (isOperator()){
144.                       return("operator");
145.                  } else if (isWhiteSpace()){
146.                       return("whitespace");
147.                       } else {
148.                            return("unknown");
149.         }
150.   }
151.   /**
152.   get a String that explains the error, if this token is an error.
153.   *
154.   @return a  String that explains the error, if this token is an
       error, null otherwise.
155.   */
156.   public String errorString(){
157.        return ("ERROR");
158.   }
159.   public Vector getTokens(){
160.        Vector v = new Vector();
161.        v.add("reservedWord");
162.        v.add("identifier");
163.        v.add("literal");
164.        v.add("operator");
165.        v.add("whitespace");
166.        v.add("unknown");
167.        return v;
168.     }
169.   }
```

- En primer lugar (líneas 5 a 21), se enumeran los tipos de token que nos podemos encontrar según las especificaciones de la gramática, codificándolas en hexadecimal para su posterior manejo. Lo hacemos así para luego identificar de forma rápida y eficiente los grupos a los que pertenecen los token mediante un simple desplazamiento de 8 bits (0x10X para las palabras reservadas, 0x20X para los identificadores, etc, donde X identifica el tipo de token dentro del

grupo). Las funciones encargadas de estos desplazamientos devuelven "True" en caso de coincidir con su correspondiente tipo (líneas 60 a 128).

- Se ha incrementado la funcionalidad de la clase añadiendo un nuevo método que nos resultara de utilidad a la hora de preguntar a la clase por los tipos de los token devueltos (líneas 159 a 168). En "MyLanguage" se devolverán seis tipos de token: "reservedWord", "identifier", "literal", "operador", "whitespace" y "unknown".

Para generar la clase hija correspondiente al Analizador **(MyLanguage)** nos ayudaremos del programa de generación de analizadores léxico JFlex.

En primer lugar especificaremos las opciones, declaraciones y código de usuario que será copiado en el comienzo del fichero fuente generado por JFlex:

```
1.  import java.io.*;
2.  import java.util.*;
3.  %%
4.  %public
5.  %class MyLanguage
6.  %implements Lexer
7.  %function getNextToken
8.  %type Token
9.  %{
10. private int lastToken;
11. private int nextState=YYINITIAL;
12. public Vector getTokens(){
13.       return (new LanguageSymbol()).getTokens();
14. }
15. public static void main(String[] args) {
16.       InputStream in;
17.       try {
18.       if (args.length > 0){
19.            File f = new File(args[0]);
20.            if (f.exists()){
21.                 if (f.canRead()){
22.                   in = new FileInputStream(f);
23.                   } else {
24.                      throw new IOException("Could not open " +
                          args[0]);
25.                   }
26.                } else {
27.                   throw new IOException("Could not find " +
                       args[0]);
28.                 }
29.          } else {
30.               in = System.in;
31.       }
32.       MyLanguage shredder = new MyLanguage(in);
33.       Token t;
34.       while ((t = shredder.getNextToken()) != null) {
```

```
35.             if (t.getID() != LanguageSymbol.WHITESPACE){
36.                 System.out.println(t);
37.             }
38.           }
39.         } catch (IOException e){
40.         System.out.println(e.getMessage());
41.       }
42.   }
43.   public void reset(java.io.Reader reader, int yyline, int yychar,
            int yycolumn) throws IOException{
44.         yyreset(reader);
45.         this.yyline = yyline;
46.         this.yychar = yychar;
47.         this.yycolumn = yycolumn;
48.   }
49.   %}
50.   %line
51.   %char
52.   %full
53.   /* main character classes */
54.   HexDigit=([0-9a-fA-F])
55.   Digit=([0-9])
56.   Letter=([a-zA-Z_$])
57.   BLANK=([ ])
58.   TAB=([\t])
59.   FF=([\f])
60.   EscChar=([\\])
61.   CR=([\r])
62.   LF=([\n])
63.   EOL=({CR}|{LF}|{CR}{LF})
64.   WhiteSpace=({BLANK}|{TAB}|{FF}|{EOL})
65.   Digit=([0-9])
66.   Letter=([a-zA-Z_$])
67.   jLetter=({Letter}|{UnicodeEscape})
68.   UnicodeEscape=({EscChar}[u]{HexDigit}{HexDigit}{HexDigit}{HexDigit
            })
69.   /* identifiers */
70.   Identifier=({jLetter}({jLetter}|{Digit})*)
71.   /* Literales enteros */
72.   DecIntegerLiteral = 0 | [1-9][Digit]*
73.   %state STRING, CHARLITERAL
74.   %%
75.   /* Palabras reservadas */
76.   <YYINITIAL> "char" {
77.   return new LanguageSymbol(LanguageSymbol.CHAR, yytext(), yyline,
            yychar, yychar+yytext().length(), nextState);
78.   }
79.   <YYINITIAL> "for" {
80.   return new LanguageSymbol(LanguageSymbol.FOR, yytext(), yyline,
            yychar, yychar+yytext().length(), nextState);
81.   }
82.   <YYINITIAL> "int" {
83.   return new LanguageSymbol(LanguageSymbol.CHAR, yytext(), yyline,
            yychar, yychar+yytext().length(), nextState);
84.   }
85.   <YYINITIAL> "if" {
86.   return new LanguageSymbol(LanguageSymbol.IF, yytext(), yyline,
            yychar, yychar+yytext().length(), nextState);
87.   }
88.   <YYINITIAL> "public" {
```

```
89.  return new LanguageSymbol(LanguageSymbol.PUBLIC, yytext(), yyline,
         yychar, yychar+yytext().length(), nextState);
90.  }
91.  <YYINITIAL> "switch" {
92.  return new LanguageSymbol(LanguageSymbol.SWITCH, yytext(), yyline,
         yychar, yychar+yytext().length(), nextState);
93.  }
94.  <YYINITIAL> "while" {
95.  return new LanguageSymbol(LanguageSymbol.WHILE, yytext(), yyline,
         yychar, yychar+yytext().length(), nextState);
96.  }
97.  /* operators */
98.  <YYINITIAL> "=" {
99.  return new LanguageSymbol(LanguageSymbol.EQ, yytext(), yyline,
         yychar, yychar+yytext().length(), nextState);
100. }
101. <YYINITIAL> "+" {
102. return new LanguageSymbol(LanguageSymbol.PLUS, yytext(), yyline,
         yychar, yychar+yytext().length(), nextState);
103. }
104. <YYINITIAL> "-" {
105. return new LanguageSymbol(LanguageSymbol.MINUS, yytext(), yyline,
         yychar, yychar+yytext().length(), nextState);
106. }
107. <YYINITIAL> "*" {
108. return new LanguageSymbol(LanguageSymbol.MULT, yytext(), yyline,
         yychar, yychar+yytext().length(), nextState);
109. }
110. <YYINITIAL> "/" {
111. return new LanguageSymbol(LanguageSymbol.DIV, yytext(), yyline,
         yychar, yychar+yytext().length(), nextState);
112. }
113. /* numeric literals */
114. <YYINITIAL> {DecIntegerLiteral} {
115. return new LanguageSymbol(LanguageSymbol.INTEGER_LITERAL,
         yytext(), yyline, yychar, yychar+yytext().length(),
         nextState);
116. }
117. /* whitespace */
118. <YYINITIAL> {WhiteSpace} {
119. return new LanguageSymbol(LanguageSymbol.WHITESPACE, yytext(),
         yyline, yychar, yychar+yytext().length(), nextState);
120. }
121. /* identifiers */
122. <YYINITIAL> {Identifier} {
123. return new LanguageSymbol(LanguageSymbol.IDENTIFIER, yytext(),
         yyline, yychar, yychar+yytext().length(), nextState);
124. }
125. <YYINITIAL> . {
126. return new LanguageSymbol(LanguageSymbol.UNKNOWN, yytext(),
         yyline, yychar, yychar+yytext().length(), YYINITIAL);
127. }
```

- El método Vector getTokens() (líneas 12 a 14), pregunta a la clase token por su vector de tipos. Este método será invocado a su vez por la herramienta y siempre deberá de ser implementado.

- El método Void main(String[] args) (líneas 15 a 42), recibe el nombre de un fichero como parámetro y pinta por pantalla los token que se reconocen.

- void reset(java.io.Reader reader, int yyline, int yychar, int yycolumn) (líneas 43 a 48), borra la actual cadena de entrada y resetea el analizador con los valores pasados como parámetros.

- Finalmente se construyen las reglas léxicas de nuestro analizador mediante las agrupaciones de los símbolos del lenguaje a reconocer y su posterior tratamiento, donde se devuelve el token reconocido (líneas 54 a 127).

5.3. Diseño de un lenguaje compatible sin JFlex.

Vamos a diseñar otro pequeño analizador léxico para que pueda ser incorporado a PlusProcessor. Este analizador realizará el resaltado de sintaxis en palabras alfabéticas iguales a 4 caracteres.

5.3.1. Implementación de un analizador para un lenguaje concreto.

La clase hija **LanguageSymbol2** es el token retornado en este lenguaje, y hereda de MatchToken. Vamos a enumerar las clases de token que nos podemos encontrar según las especificaciones de la gramática y a codificarlas en hexadecimal para su manejo. En este caso existen las siguientes:

public final static int CUATRO = 0x101;
public final static int WHITESPACE = 0xE00;
public final static int UNKNOWN = 0xF00;

Igual que en el caso anterior, deberemos definir los métodos heredados de la clase padre para que puedan ser manejados por nuestro analizador. La diferencia con el anterior ejemplo estriba particularmente en que ahora son tres las clases de token que serán devueltos. Como ejemplo detallaremos la función getTokens, donde se devuelven los tres tipos de token que se analizan en el lenguaje.

```
public Vector getTokens(){
            Vector v = new Vector();
            v.add("reservedWord");
            v.add("whitespace");
            v.add("unknown");
            return v;
        }
```

Creamos la clase hija correspondiente al Analizador **(MyLanguage2)** directamente en Java:

```
1.  package MyLanguage2;
2.  import languageplugin.Lexer;
3.  import languageplugin.Token;
4.  import java.io.*;
5.  import java.util.*;
6.  public class MyLanguage2 implements Lexer {
7.  private static final char BLANK=' ';
8.  private static final char FF='\f';
9.  private static final char TAB='\t';
10. private static final char CR='\r';
11. private static final char LF='\n';
12. private static final char CRLF=CR+LF;
13. /** the input device */
14. private java.io.Reader zzReader;
15. private int dato = 0;
16. /** number of newlines encountered up to the start of the matched
          text */
17. private int yyline = 0;
18. /** the number of characters up to the start of the matched text
          */
19. private int yychar = 0;
20. public Vector getTokens(){
21.       return (new LanguageSymbol2()).getTokens();
22. }
23. public void reset(java.io.Reader reader, int yyline, int yychar,
          int yycolumn) throws IOException{
24.       yyreset(reader);
25.       this.yyline = yyline;
26.       this.yychar = yychar;
27. }
28. public MyLanguage2(java.io.Reader in) {
29.       this.zzReader = in;
30. }
31. /**
32. Closes the input stream.
33. */
34. public final void yyclose() throws java.io.IOException {
35.       if (zzReader != null)
36.       zzReader.close();
37. }
38. public final void yyreset(java.io.Reader reader) {
39.       zzReader = reader;
40.       yyline = yychar  = 0;
41. }
42. public Token getNextToken() throws java.io.IOException {
43.       int startRead=yychar;
44.       dato=zzReader.read();
45.       yychar++;
46.       while  ((dato  !=  -1)&&  ((dato  ==  TAB)  ||  (dato  ==
          BLANK)||(dato == FF)||(dato == CR)||(dato == LF)||(dato
          == CRLF)))
47.       {
48.            dato=zzReader.read();
49.            yychar++;
50.       }
```

```
51.       if (((char)dato >= 'A' && (char)dato <= 'Z')||((char)dato >=
              'a' && (char)dato <= 'z'))
52.       {
53.             dato=zzReader.read();
54.             yychar++;
55.             if (((char)dato >= 'A' && (char)dato <=
                  'Z')||((char)dato >= 'a' && (char)dato <= 'z'))
56.             {
57.                   dato=zzReader.read();
58.                   yychar++;
59.                   if (((char)dato >= 'A' && (char)dato <=
                    'Z')||((char)dato >= 'a' && (char)dato <= 'z'))
60.                   {
61.                     dato=zzReader.read();
62.                     yychar++;
63.                     if (((char)dato >= 'A' && (char)dato <=
                        'Z')||((char)dato >= 'a' && (char)dato <=
                        'z'))
64.                     {
65.                        dato=zzReader.read();
66.                        yychar++;
67.                        if ((dato == BLANK)||(dato == FF)||(dato ==
                      CR)||(dato == LF)||(dato == CRLF))
68.                        {
69.                              return new LanguageSymbol2
                            (LanguageSymbol2.CUATRO, "",yyline,
                            startRead, yychar-1, -1);
70.                        }
71.                        else
72.                        {
73.                             while ((dato != -1)&&(dato !=
                           BLANK)&&(dato != FF)&&(dato !=
                      CR)&&(dato != LF)&& (dato != TAB))
74.                             {
75.                               if (dato == LF)
76.                                  yyline++;
77.                               dato=zzReader.read();
78.                               yychar++;
79.                             }
80.                        return new LanguageSymbol2
                       (LanguageSymbol2.UNKNOWN, "", yyline,
                    startRead, yychar-1, -1);
81.                     }
82.                   }
83.             }
84.       }
85.       }
86.       while ((dato != -1)&&(dato != BLANK)&&(dato != FF)&&(dato !=
                CR)&&(dato != LF))
87.       {
88.       if (dato == LF)
89.            yyline++;
90.       dato=zzReader.read();
91.       yychar++;
92.       }
93.       if (dato != -1)
94.            return new LanguageSymbol2(LanguageSymbol2.UNKNOWN, "",
              yyline, startRead, yychar-1, -1);
95.       else
96.            return null;
97.       }
```

```
98.  }
```

- Declaramos el método principal de la clase que, como podemos ver, implementa a la clase abstracta Lexer (línea 6).

- Se declaran los caracteres especiales que nos podemos encontrar y que debemos de alguna forma ignorar como blancos, tabuladores, saltos de línea, etc (líneas 7 a 10).

- Tipo de entrada y dato donde se leerán los caracteres (líneas 14 y 15.

- Variable para contar el número de nuevas líneas encontradas hasta el comienzo del token encontrado (línea 17).

- Variable para contra el número de caracteres leídos hasta el comienzo del token encontrado (línea 19).

- Método mediante el cual se le puede preguntar a la clase Token sobre su vector de tipos. Este método es invocado por la herramienta y es necesario declararlo en todos los analizadores léxicos que deseemos implementar (líneas 20 a la 22).

- Método para resetear los contadores del analizador (líneas 23 a 27).

- Método principal que se invoca cuando creamos una clase de este tipo. Hay que pasarle como argumento el texto a analizar en formato Reader (líneas 28 a 30).

- Método para cerrar la cadena de entrada (líneas 34 a 37).

- Método para resetear la cadena de entrada y el número de líneas leídas (líneas 38 a 41).

- Método para leer los token. Es obligatorio especificarlo, ya que es utilizado por la herramienta. Su importancia es determinante para analizar los token que se

van tratando por la cadena de entrada (líneas 42 a 97). Pasamos a describir con más detalle por la importancia de este método:

-Se tiene en cuenta desde que parte del texto se comienza a leer mediante la variable startRead (línea 43).

-Se lee un carácter en la variable dato y se incrementa el contador de caracteres generales leídos en el documento (líneas 44 y 45).

-Mientras no se haya llegado al final del fichero y el dato no sea un carácter de interés, leemos de la entrada e incrementamos el contador de caracteres leídos (líneas 46 a 50).

-Si los cuatro caracteres siguientes son letras y el quinto carácter es un "blanco", podremos decir que hemos encontrado un token de cuatro caracteres alfabéticos de longitud, por tanto, se habrá detectado que existe un token . Se crea un nuevo objeto token de tipo "Cuatro" y se devuelve a la aplicación que invocó el método de la clase. En otro caso, si no se ha llegado al final del fichero y hay más de cuatro caracteres cualesquiera, se debe de retornar un token "desconocido" inmediatamente después de detectar un blanco. Cuando se llega al final del fichero (dato = -1) es importante retornar **null** para que la herramienta tenga constancia de que se ha terminado de analizar la cadena de entrada, de lo contrario, se producirá un comportamiento no deseado en el funcionamiento del analizador (líneas 51 a 97).

5.4. Creación de un plugin para PlusProcessor.

Una vez que hemos creado las dos clases necesarias para un analizador léxico determinado tenemos que integrarlo en la herramienta como un plugin. Para ello debemos empaquetar en un fichero .jar los siguientes componentes:

- Fichero .class generado para la clase token
- Fichero .class generado para la clase lexer

- Fichero Manifest (.mf) que deberá incluir una propiedad llamada Language-Class-Name, donde figurará el nombre del paquete que contiene estas clases, seguido de un punto y el nombre de la clase principal encargada de analizar lexicográficamente. Un ejemplo del contenido del Manifest.mf podría ser el siguiente:

```
Manifest-Version: 1.0
Language-Class-Name: PackageName.LanguageName
```

Una vez realizadas estas tareas, solamente nos quedará incorporar el fichero .jar generado en la carpeta **plugins** de la herramienta, que será cargado en la próxima ejecución de la misma.

Capítulo 6

ESPECIFICACIÓN DE REQUISITOS

Desde el inicio del desarrollo de sistemas, los ingenieros nos hemos topado con un gran problema, la identificación de los requisitos del sistema. Esto es debido a que no es un proceso que pueda ser determinado matemáticamente. Es un proceso en el cual los datos son extraídos de las personas y estos datos pueden variar, dependiendo de la persona a la cual estemos consultando. Es por eso que la Ingeniería de Requisitos ha trabajado arduamente para tratar de desarrollar técnicas que permitan hacer este proceso de una forma más eficiente y segura.

Algunas de la técnicas utilizadas para la captura de requisitos son las siguientes:

- **Introspección:** esta técnica recomienda que el ingeniero de requisitos se ponga en el lugar del cliente y trate de imaginar como desearía él el Sistema. Y en base a estas suposiciones comenzar a recomendar al cliente sobre la funcionalidad que debería presentar el sistema. El problema radica en que un ingeniero no es un tipo normal de cliente, posee un conocimiento técnico más elevado por lo que se podrían recomendar cosas que el Cliente no necesite.
- **Análisis de Protocolo:** esta técnica parte de la idea de que el cliente cuenta con un modelo mental preexistente del sistema deseado y en base a este modelo ya existente se puede analizar y obtener los requisitos del sistema. Es una técnica muy poco utilizada debido a que los Clientes rara vez poseen una idea clara de lo que desean en su sistema.
- **Casos de Uso:** Es una técnica bastante utilizada que captura cada una de las funciones del sistema y en base a cada una de ellas especifica los requisitos del mismo.
- **VORD:** Esta técnica es utilizada para capturar requisitos en base a puntos de vista. Es utilizado en sistemas que van a ser desarrollados con el paradigma de programación orientados a objetos.

En este proyecto se han estudiado algunas de las herramientas más representativas de gestión de software, algunas de ellas están enfocadas a la visualización del software

y otras son herramientas educativas. No existen clientes o usuarios con los que reunirse para la captura de requisitos, si no que los requisitos se obtuvieron partiendo de dichas herramientas.

En cuanto a los requisitos a considerar se puede distinguir a grandes rasgos entre: *funcionales*, aquellos que especifican las acciones que debe ser capaz de realizar el sistema; y *no funcionales*, aquellos que especifican propiedades del sistema.

El conjunto total de requisitos se ha tenido en cuenta al definir la arquitectura de la API, de forma que su posterior ampliación para incorporarlos no sea costosa. Como indica el proceso de desarrollo de programación extrema, durante el desarrollo de la API han ido apareciendo nuevos requisitos sobre nuevas necesidades, por lo que aquí se muestran tanto los iniciales como los que han ido surgiendo [2].

En los siguientes apartados se muestran los requisitos en un formato inspirado en el estándar IEEE 830-1998.

Una información más completa sobre la especificación de requisitos se encuentra en [2].

6.1 Descripción general.

6.1.1 Perspectiva del producto.

Interfaces del sistema

La definición de la API es totalmente independiente a cualquier otra parte del sistema. No obstante, las implementaciones posiblemente dependan de otras partes del sistema para su correcto funcionamiento.

Interfaces de usuario

La API debe proporcionar un componente de texto para la edición de texto, este es el único requisito en el que se menciona el interfaz de usuario. Este componente textual debe estar definido usando el estándar para la construcción de interfaces gráficos en Java, Swing.

Interfaces software

La API se definirá usando la tecnología Java. No se consideran requisitos software adicionales a los que impone la propia tecnología Java. No obstante, las diferentes implementaciones de la API pueden considerar como requisito la existencia de cierto software.

6.1.2 Funciones del producto.

Las principales funciones de la API son las de gestionar en todos sus aspectos un Fichero de texto escrito en cualquier lenguaje y facilitar las herramientas adecuadas para su gestión.

6.1.3 Características del usuario.

Los usuarios que deseen incorporar plugin a la API deberán conocer los principios básicos de construcción de analizadores de lenguajes. El resto de usuarios no necesitan ningún conocimiento adicional para la edición de textos.

6.2 Requisitos específicos.

6.2.1 Interfaces Externos.

La salida de la API serán programas escritos en el lenguaje Java en un formato ejecutable.

6.2.2 Requisitos funcionales.

Se van a especificar todos los requisitos relativos a la gestión de los elementos que forman la definición del programa:

R1. La herramienta facilitará un menú, requerido e integrable por el usuario, donde se podrán utilizar las funcionalidades para ficheros, edición, estilo y configuración.

R2. Se incorporarán las funcionalidades de *apertura y nueva creación* de ficheros, que serán utilizadas desde el menú o mediante la invocación de

métodos. Estas opciones podrán también estar ocultas a petición del usuario del menú.

R3. Se incorporarán al menú las funcionalidades de *salvar, salvar como y salvar todos* los ficheros manipulados en un instante determinado. Estas opciones podrán también estar ocultas a petición del usuario del menú.

R4. Se podrán utilizar las opciones de *salvar* y *salvar como* mediante un menú, que será visualizado al pulsar el botón derecho del ratón en el documento seleccionado.

R5. Se incorporarán al menú las funcionalidades de *cerrar, y cerrar todos* los ficheros manipulados en un instante determinado. Estas opciones podrán también estar ocultas a petición del usuario del menú.

R6. Se podrán utilizar las opciones de *cerrar y cerrar todos* los ficheros mediante un menú, que será visualizado al pulsar el botón derecho del ratón en el documento seleccionado.

R7. Se incorporará al menú la opción de *salir* de la aplicación en el menú aportado por la herramienta. Esta opción podrá también estar oculta a petición del usuario del menú.

R8. Se incorporarán al menú las funcionalidades *deshacer y rehacer* escritura en un documento determinado. Estas opciones podrán también estar ocultas a petición del usuario del menú.

R9. Se incorporarán al menú las funcionalidades *cortar, pegar y copiar* escritura entre todos los documentos manipulados. Estas opciones podrán también estar ocultas a petición del usuario del menú.

R10. Se podrán utilizar las opciones de *cortar, pegar y copiar* escritura entre todos los documentos manipulados mediante un menú, que será visualizado al pulsar el botón derecho del ratón en el documento seleccionado.

R11. Se incorporará al menú la posibilidad de poder *seleccionar todo* el texto de un documento determinado que además podrá ser requerido por el usuario.

R12. Se incorporarán al menú la posibilidad de poder *seleccionar todo* el texto de un documento determinado mediante un menú, que será visualizado al pulsar el botón derecho del ratón en el documento seleccionado.

R13. Se podrán realizar mediante opciones de menú búsquedas de cadenas de texto en ambos sentidos del documento (siguiente coincidencia y todas las coincidencias). Además se podrá determinar el alcance (líneas seleccionadas o todo el documento) y las opciones oportunas. Estas opciones podrán también estar ocultas a petición del usuario del menú.

R14. Se podrán realizar mediante opciones de menú reemplazamientos de cadenas de texto en ambos sentidos del documento (reemplazar la siguiente coincidencia y reemplazar todas las coincidencias). Además se podrá determinar el alcance (líneas seleccionadas o todo el documento) y las opciones oportunas. Estas opciones podrán también estar ocultas a petición del usuario del menú.

R15. La herramienta podrá permitir mediante opciones de menú *aumentar o disminuir* el tamaño del texto de los documentos manipulados. Estas opciones podrán también estar ocultas a petición del usuario del menú.

R16. La herramienta permitirá la configuración del analizador seleccionado para cada documento que se esta manipulando. También permitirá el cambio de analizador para dichos documento de una forma inmediata mediante componentes `RadioButtom`.

R17. La herramienta dispondrá de una serie de preferencias para controlar los estilos de texto de cada uno de los patrones usados para el coloreado de sintaxis (tipo de fuente, color de cada token, cursiva y negrita).

R18. Los cambios realizados en los patrones podrán ser guardados y recuperados al término e inicio de la utilización de la herramienta.

R19. Los cambios realizados en los documentos que sean de carácter general, podrán ser guardados y recuperados al término e inicio de la utilización de la herramienta.

R20. Se podrá permitir al usuario de la API la posibilidad de marcado y borrado de marcado de palabras que coincidan con determinada cadena.

R21. Se podrá permitir al usuario de la aplicación que incorpore la API la posibilidad de marcado y borrado de marcado de un texto continuo dentro de un documento.

R22. Se deberá crear un componente que permita abrir varios ficheros a la vez.

R23. Cuando la opción elegida por el usuario de la API sea la de pedir a la herramienta un componente para incluir varios documentos, se deberá distinguir de forma clara qué documentos modificados no están salvados en disco.

R24. Incorporará funcionalidades para visualizar y ocultar una barra en la que aparecerán todas las marcas del documento y pulsando en ellas se irá a la zona en la que se encuentre.

R25. Se deberá de permitir abrir ficheros individuales para su uso de forma independiente.

R26. La herramienta deberá tener un sistema de plugins que permita incorporar nuevos lenguajes a la misma.

R27. El usuario de la API podrá especificar el analizador léxico utilizado por defecto para abrir los documentos.

R28. Todas las indicaciones por parte de la herramienta hacia el usuario de la aplicación que incorpore la API para informar de indeseados estados de ejecución, deberán realizarse a través de cajas de dialogo.

6.2.3 Requisitos no funcionales.

Una vez especificados los requisitos funcionales del sistema, se especifican los no funcionales:

- **REQNF1 Fiabilidad:** No existen atributos del sistema para este apartado.
- **REQNF2 Disponibilidad:** No existen atributos del sistema para este apartado.
- **REQNF3 Mantenibilidad:** La API debe ser suficientemente flexible para que se pueda mantener y mejorar de forma sencilla.
- **REQNF4 Portabilidad:** Esta API debe ser portable a cualquier plataforma que soporte la tecnología Java.
- **REQNF5 Tiempo de respuesta:** No existen atributos del sistema para este apartado.
- **REQNF6 Utilización de memoria:** No existen atributos del sistema para este apartado.

6.3 Conclusiones.

En este Proyecto Fin de Carrera se definirá la API con todos los requisitos enumerados anteriormente y se desarrollará una implementación por defecto de los mismos. La ampliación de la API, de forma que incorpore nuevos requisitos y el desarrollo de implementaciones para ellos, será llevada a cabo en posteriores Proyectos.

Capítulo 7

ARQUITECTURA

7.1. Introducción a la arquitectura.

Una Arquitectura es un conjunto organizado de elementos y el diseño de más alto nivel de la estructura de un sistema, programa o aplicación. Se utiliza para especificar las decisiones estratégicas acerca de la estructura y funcionalidad del sistema, las colaboraciones entre sus distintos elementos y su despliegue físico para cumplir unas responsabilidades bien definidas. Tiene la responsabilidad de:

- Definir los módulos principales
- Definir las responsabilidades que tendrá cada uno de estos módulos
- Definir la interacción que existirá entre dichos módulos:
    - o Control y flujo de datos.
    - o Secuenciación de la información.
    - o Protocolos de interacción y comunicación.
    - o Ubicación en el hardware.

La Arquitectura del sistema aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.

La definición oficial de Arquitectura es la IEEE Std 1471-2000 que reza así: "*La Arquitectura es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución*".

El objetivo principal de la Arquitectura del sistema es aportar elementos que ayuden a la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participen en un

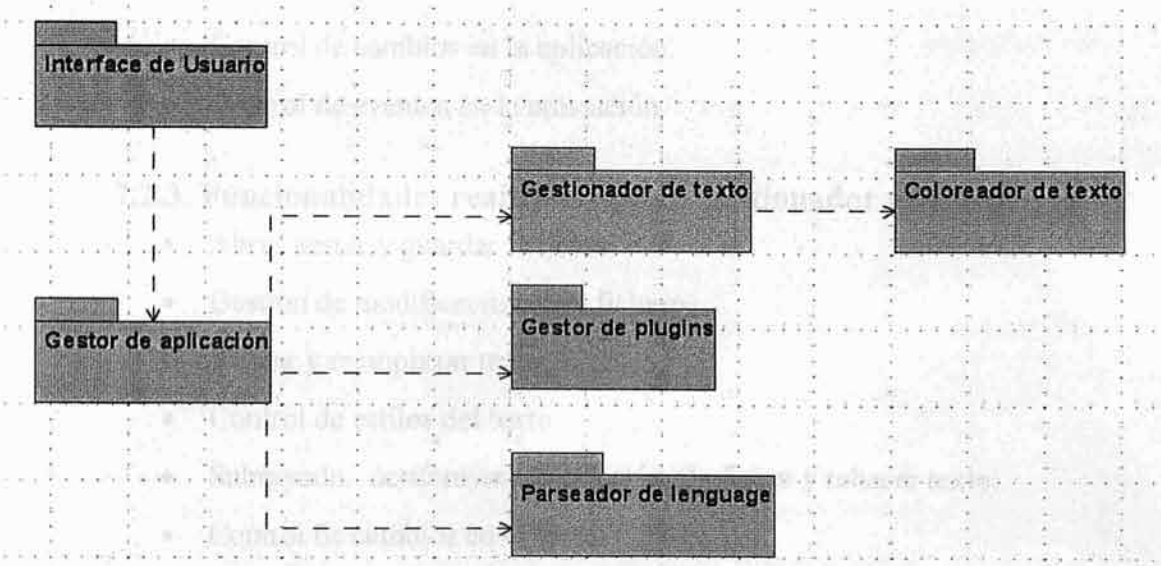
proyecto. Para conseguirlo, la Arquitectura construye abstracciones, materializándolas en forma de diagramas.

Se va a utilizar el Lenguaje Unificado de Desarrollo (UML) para mostrar la arquitectura del proyecto, ya que existe la necesidad de organizar dichas abstracciones en vistas, tal y como se hace al diseñar un edificio. La cantidad y tipos de vistas difiere en función de cada tendencia arquitectónica.

Una información más completa sobre la introducción a la arquitectura se encuentra en [1] y [9].

7.2. Arquitectura del proyecto.

En este proyecto se han organizado los componentes de la arquitectura según sus funcionalidades. Su estructura general es la siguiente:



7.2.1. Funcionalidades realizadas por el interface de usuario.

- Representación grafica de las herramientas proporcionadas por la aplicación.
- Inter-actuación e interpretación de eventos provocados por el usuario final.
- Representación de cuadros de dialogo.

7.2.2. Funcionalidades realizadas por el gestor de aplicación.

- Distribución de tareas.
- Inicialización de los estados de la aplicación.
- Cierre ordenado de la aplicación.
- Facilitar una interface para la gestión externa de la herramienta como librería.
- Control de cambios en la aplicación.
- Control de eventos en la aplicación.

7.2.3. Funcionalidades realizadas por el gestionador de texto.

- Abrir, cerrar y guardar ficheros.
- Gestión de modificaciones en ficheros.
- Buscar y reemplazar texto.
- Control de estilos del texto.
- Subrayado, desubrayado, selección, deshacer y rehacer texto.
- Control de cambios en el texto.

7.2.4. Funcionalidades realizadas por el coloreador de texto.

- Coloreado de la sintaxis.

7.2.5. Funcionalidades realizadas por el gestor de plugins.

- Carga de nuevos plugins.
- Instanciación de nuevos objetos para agregar funcionalidades.

7.2.6. Funcionalidades realizadas por el parseador del lenguaje.

- Análisis lexicográfico del texto.
- Devolución de los tokens reconocidos.

Capítulo 8

DISEÑO

8.1. Descripción de las clases.

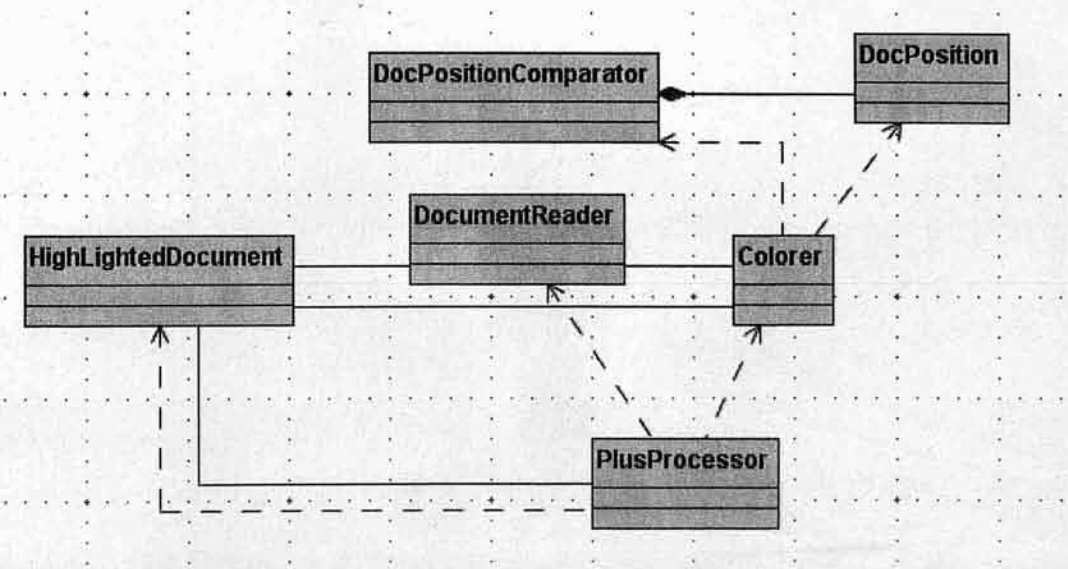
Se van a describir todas las clases creadas para realizar la herramienta. Irán acompañadas de un diagrama de clases que contemplarán todas las descripciones realizadas.

8.1.1. Paquete PlusProcessor.

- **Colorer:** Su función es la de colorear el documento activo con los estilos configurados para el analizador léxico elegido. Cada uno de los documentos tiene asignado un Colorer que vigila, en forma de tarea, los cambios producidos en el mismo, coloreando con su estilo correspondiente la palabra cambiada.

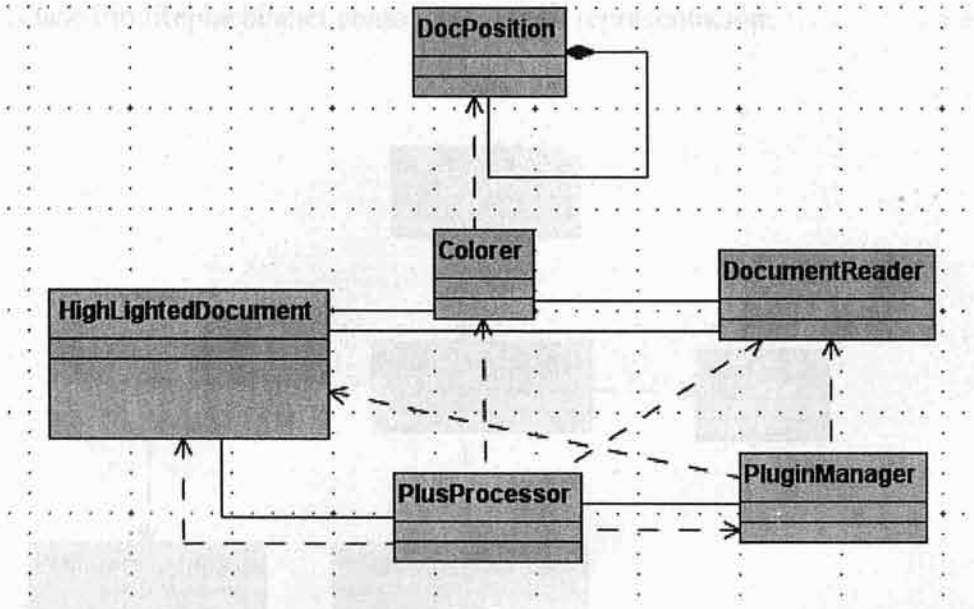
- **DocPosition:** Se utiliza como envoltorio para almacenar una posición en cierto documento y poder ser almacenado en una colección.

- **DocPositionComparator:** Se utiliza para comparar objetos de tipo DocPosition (igual, mayor y menor).



- **DocumentReader:** Es un interface de lectura para un documento abstracto utilizado por defecto. Se utiliza para leer, saltar, resetear y constatar el puntero de lectura en dicho documento.

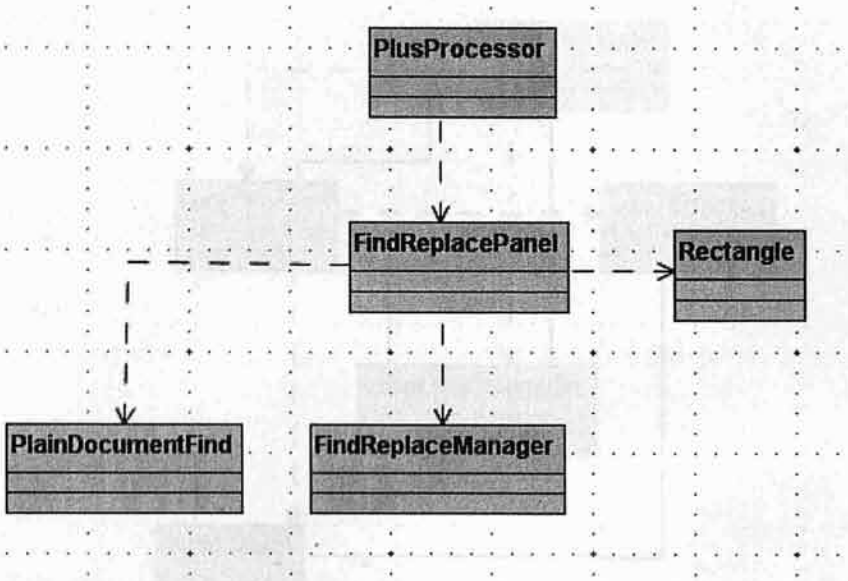
- **HighLightedDocument:** Es un documento DefaultStyledDocument que se le han añadido métodos para interceptar las modificaciones y las preparlas para su posterior coloreado.



- **FindReplaceManager:** Es el interface de gestión para la búsqueda y reemplazo de cadenas de caracteres en un determinado documento.

- **FindReplacePanel:** Se encarga de representar el interfaz de interpretación e inter-actuación de usuario para la búsqueda y reemplazo de cadenas de caracteres en un determinado documento.

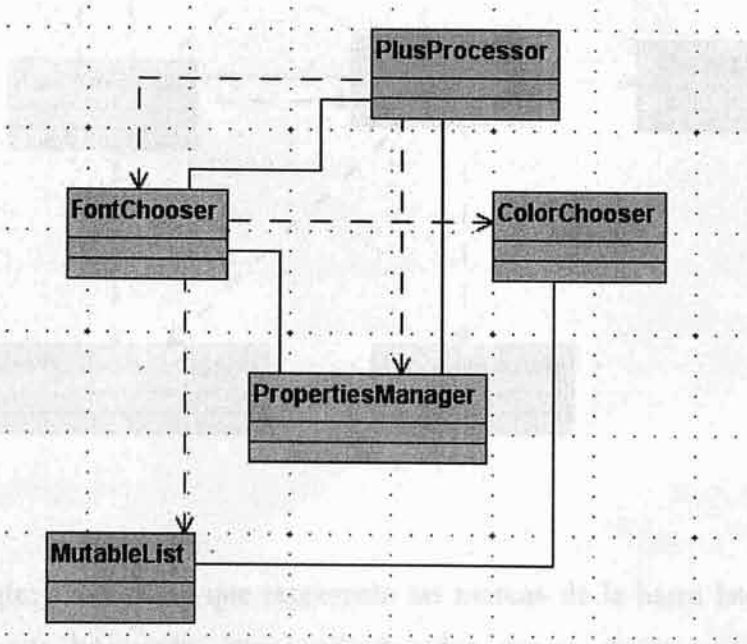
- **PlainDocumentFind:** Es un documento PlainDocument que se le han añadido ciertos métodos para modificar e insertar texto. Es utilizado por la clase FindReplacePanel como motor de su representación.



- **FontChooser:** Esta clase se encarga de representar el interfaz de interpretación e inter-actuación de usuario para cambiar las propiedades de los token en un determinado analizador.

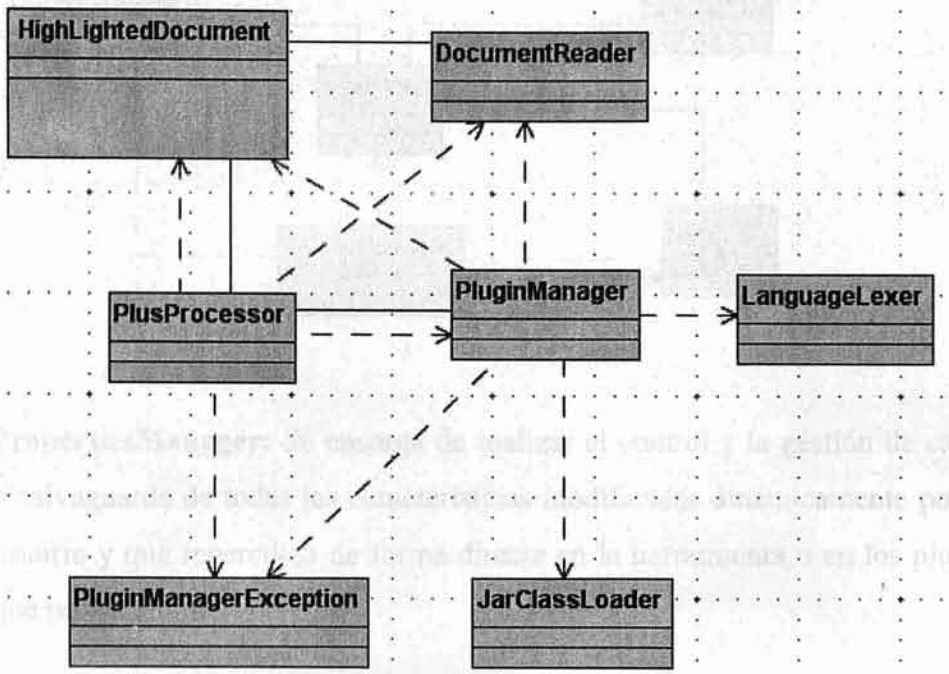
- **ColorChooser:** Se encarga de obtener y cambiar un determinado color a ciertos elementos a partir de un determinado estilo.

- **MutableList:** Es utilizada por FontChooser para llevar el control de los estilos que el usuario cambia para determinados tokens de un analizador. Mediante esta clase se podrá retomar la situación inicial en caso que el usuario decida cancelar la operación.



- **PluginManager:** Se encarga de realizar todas las gestiones para cargar los plugins existentes para la herramienta. Otra de sus tareas es la instanciación de las nuevas clases agregadas para su posterior uso.

- **PluginManagerException:** Es un manejador de excepciones para una mayor versatilidad del código.

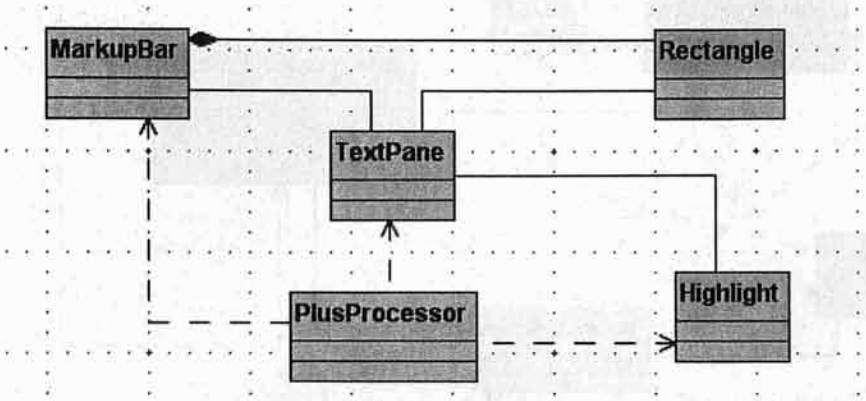


- **Rectangle:** Es la clase que representa las marcas de la barra lateral en una determinada búsqueda, cuyas coordenadas dentro de la misma quedan registradas cada vez que se detecte la existencia de una cadena de caracteres que coincida con un determinado patrón deseado.

- **MarkupBar:** Se encarga de la construcción, borrado, modificación y manejo de la barra de marcas lateral usada para representar todas las coincidencias de una determinada búsqueda.

- **TextPane:** Es un documento JTextPane que se le han atributos y métodos para almacenar los datos y gestionar los estados de un determinado documento abierto por la herramienta.

- **Highlight:** Se encarga de subrayar y desubrayar palabras o cierta longitud de texto en un determinado panel de texto.



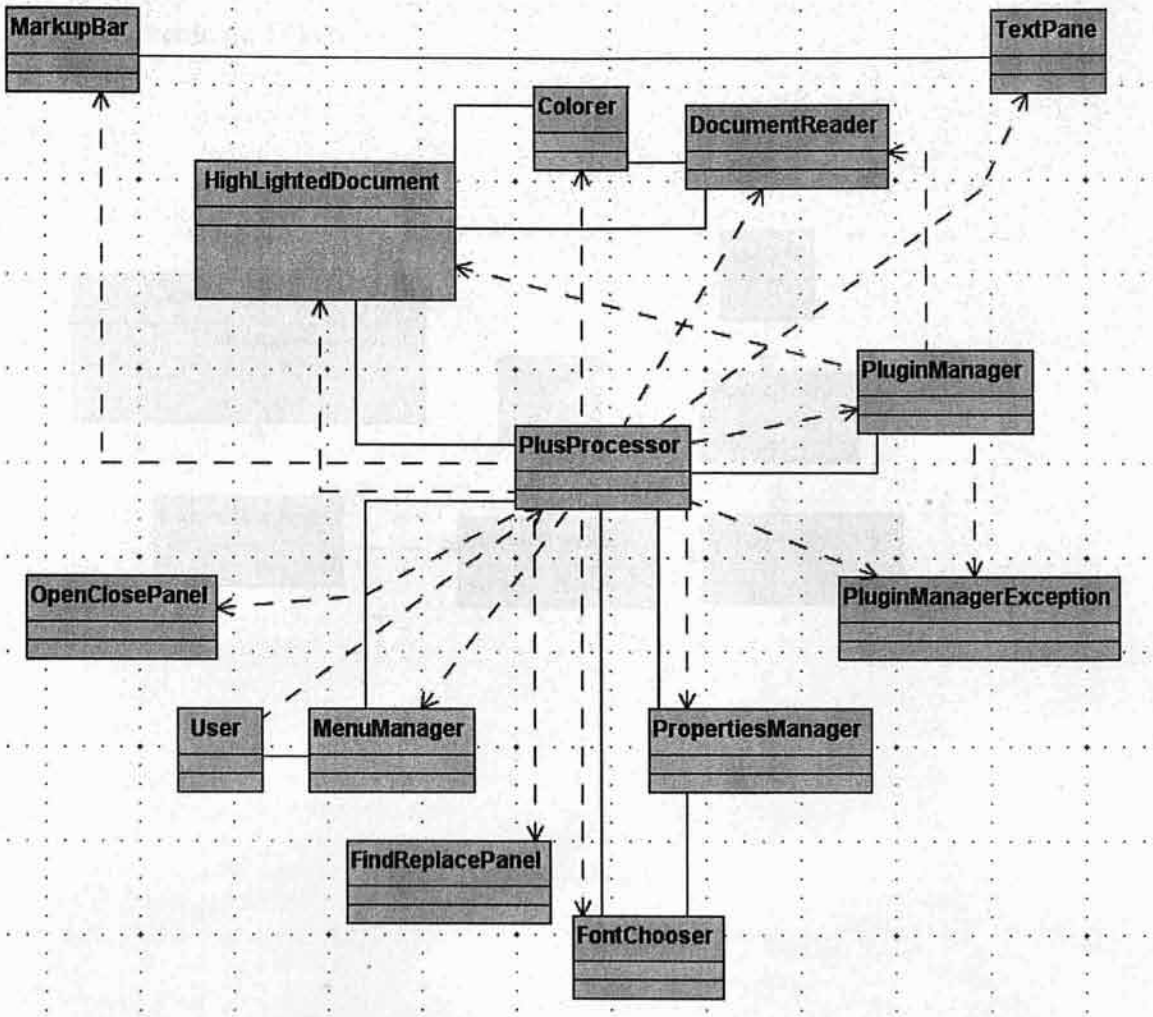
- **PropertiesManager:** Se encarga de realizar el control y la gestión de carga o salvaguarda de todas las características modificadas dinámicamente por el usuario y que repercuten de forma directa en la herramienta o en los plugin que proporciona.

- **OpenClosePanel:** Se trata de una clase que gestiona la apertura, salvaguarda y cierre los ficheros abiertos por la herramienta. Proporciona una interface de ínter-actuación con el usuario final.

- **MenuManager:** Es el interface de menús de la herramienta encargada de canalizar los eventos provocados por la ínter-actuación del usuario. Además le podrá proporcionar diversas funcionalidades para el manejo de los submenus.

- **PlusProcessor:** Es la Clase principal de la herramienta y es la que proporciona al Usuario los métodos adecuados para utilizar todas las funcionalidades para las cuales se ha creado dicha herramienta. Su misión es la de repartir y gestionar las funcionalidades a las principales clases, tanto en

el inicio y cierre de la aplicación como en la nueva creación de los contenedores de uno o varios ficheros de texto y de sus posibles estados. Es el interfaz de usuario propiamente dicho.



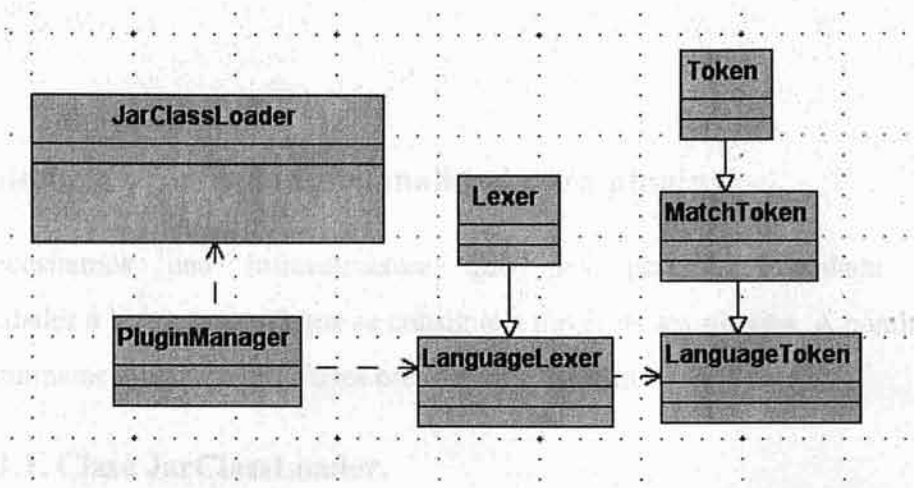
8.1.2. Paquete LanguajePlugin.

- **JarClassLoader:** Se encarga de definir y resolver un fichero .class empaquetado dentro de un .jar para a fin de poder devolver una clase apta para su instanciación dentro del entorno de la herramienta.

- **Lexer:** Es el interfaz genérico que deberá implementarse cuando deseemos crear un analizador léxico.

- **Token:** Es el interfaz genérico que deberá implementarse cuando deseemos crear los tokens de un determinado analizador léxico.

- **MatchToken:** Clase abstracta suministrada por la herramienta para tener una referencia en la implementación de una clase Token. Esta clase hereda directamente de Token.



Capítulo 9

IMPLEMENTACIÓN

Vamos a describir la implementación de dos de las funcionalidades aportadas por la herramienta por entender que representan una parte relevante en el conjunto de la misma.

9.1. Implementación de la funcionalidad para plugins.

Necesitamos una infraestructura que nos permita introducir nuevas funcionalidades a la herramienta que se consigue a través de los plugins. A continuación se van a enumerar y explicar las clases creadas para este fin.

9.1.1. Clase JarClassLoader.

```
1. package languageplugin;
2. import java.io.File;
3. import java.io.IOException;
4. import java.io.InputStream;
5. import java.net.MalformedURLException;
6. import java.net.URL;
7. import java.util.ArrayList;
8. import java.util.Enumeration;
9. import java.util.HashMap;
10. import java.util.Hashtable;
11. import java.util.List;
12. import java.util.Map;
13. import java.util.jar.JarEntry;
14. import java.util.jar.JarFile;
15. public class JarClassLoader extends ClassLoader {
16. private Map<String, Class> cache = new HashMap<String, Class>();
17. static final int BUFFER_SIZE = 2048;
18. private File file;
19. private String baseString;
20. public JarClassLoader(File file) throws IOException {
21.       this.file = file;
22.       // Formato explicado en java.net.JarURLConnection
23.       this.baseString                =                "jar:file:/"+
          file.getAbsolutePath().replace('\\', '/') + "!/";
24.       loadClassNames();
25. }
26. private void loadClassNames() throws IOException {
27.      JarFile jarFile = new JarFile(this.file);
28.      Enumeration<JarEntry> e = jarFile.entries();
```

```
29.       while (e.hasMoreElements()) {
30.            JarEntry je = e.nextElement();
31.            String name = je.getName();
32.            if (name.endsWith(".class")) {
33.                 name = name.substring(0, name.length() - 6);
34.                 name = name.replace('/', '.');
35.                 cache.put(name, null);
36.            }
37.       }
38. }
39. //---------- Abstract Implementation --------------------
40. public synchronized Class loadClass(String className)
41. throws ClassNotFoundException {
42.       if (classInJar(className)) {
43.            Class clazz = (Class) cache.get(className);
44.            if (clazz != null) {
45.                 return clazz;
46.            }
47.            String nombreFichero = formatClassName(className);
48.            InputStream is;
49.            try {
50.            is = new URL(baseString + nombreFichero).openStream();
51.            } catch (IOException e) {
52.            throw new ClassNotFoundException(className, e);
53.       }
54.       byte[] bytes = readBytes(is);
55.       clazz = defineClass(className, bytes, 0, bytes.length);
56.       resolveClass(clazz);
57.       cache.put(className, clazz);
58.       return clazz;
59.       } else {
60.            return
    ClassLoader.getSystemClassLoader().loadClass(className);
61.       }
62. }
63. /**
64. */
65. private boolean classInJar(String className) {
66.       return this.cache.containsKey(className);
67. }
68. protected String formatClassName(String className) {
69.       return className.replace('.', '/') + ".class";
70. }
71. /**
72. */
73. private byte[] readBytes(InputStream in) {
74.       try {
75.            List bytesList = new ArrayList();
76.            int size = 0;
77.            byte[] bytes = new byte[BUFFER_SIZE];
78.            int leidos = in.read(bytes);
79.            while (leidos != -1) {
80.                 if (leidos != BUFFER_SIZE) {
81.                 byte[] aux = new byte[leidos];
82.                 System.arraycopy(bytes, 0, aux, 0, leidos);
83.                 bytesList.add(aux);
84.                 } else {
85.                      bytesList.add(bytes);
86.                      bytes = new byte[BUFFER_SIZE];
87.                 }
88.                 size += leidos;
```

```
89.                 leidos = in.read(bytes);
90.             }
91.             byte[] nuevosBytes = new byte[size];
92.             int contador = 0;
93.             for (int i = 0; i < bytesList.size(); i++) {
94.                 bytes = (byte[]) bytesList.get(i);
95.                 System.arraycopy(bytes, 0, nuevosBytes,
                    contador, bytes.length);
96.                 contador += bytes.length;
97.             }
98.             return nuevosBytes;
99.             } catch (IOException e) {
100.                 return null;
101.             }
102.         }
103.     public File getFile() {
104.         return this.file;
105.     }
106. }
```

- JarClassLoader se crea para cargar clases desde un fichero jar. Declarar un mapa de dispersión para guardar la información de las clases y un tamaño para el buffer de lectura. (línea 16 a 17).

- Se analiza el camino absoluto para llegar al fichero jar a chequear (línea 23) y se extraen los nombres de las clases que contiene dicho fichero. A continuación la introducimos sus nombres en un hash para su posterior comprobación cuando necesiten ser cargadas. (línea 26 a 38).

- El método loadClass se encarga de comprobar que la clase se encuentra cargada en la tabla hash (se cargan bajo demanda). Si es así, se devuelve la clase correspondiente. De lo contrario, se leen los bytes de dicha clase (método readBytes), se valida el total de bytes leidos para conformar la clase, se introduce en la tabla hash y se retorna dicha clase apta para su instanciación (línea 40 a 62).

9.1.2. Clase PluginManager.

```
1. package PlusProcessor;
2. import java.io.File;
3. import java.io.FilenameFilter;
4. import java.io.IOException;
5. import java.io.Reader;
6. import java.lang.reflect.Constructor;
```

```
7.  import java.lang.reflect.InvocationTargetException;
8.  import java.util.Vector;
9.  import java.util.jar.JarFile;
10. import javax.swing.JOptionPane;
11. import languageplugin.JarClassLoader;
12. import languageplugin.Lexer;
13. public class PluginManager{
14. private Vector vPlugins;
15. private Vector vToken, vTokens;
16. private int plugins = 0;
17. private PlusProcessor plusProcessor;
18. public PluginManager(PlusProcessor pProcessor){
19.       plusProcessor = pProcessor;
20.       vPlugins = new Vector();
21.       File[]    files    =    new    File("plugins").listFiles(new
          FilenameFilter() {
22.           public boolean accept(File dir, String name) {
23.                return name.endsWith(".jar");
24.           }
25.       });
26.       vPlugins.add(plugins, null);
27.       vToken = new Vector();
28.       vToken.add(0, "DefaultLexer");
29.       plusProcessor.setFontChoosers(plugins, "Default-Token",
           vToken);
30.       plugins++;
31.       vTokens = new Vector();
32.       vTokens.add(0, vToken);
33.       for (File file : files) {
34.       try {
35.            JarFile jf = new JarFile(file);
36.            String languageName = (String) jf.getManifest()
               .getMainAttributes().getValue("Language-Class-Name");
37.            JarClassLoader jcl = new JarClassLoader(file);
38.           Class c;
39.           c = jcl.loadClass(languageName);
40.           vPlugins.add(plugins, c);
41.           if (languageName.indexOf(".")>=0 )
42.              languageName=languageName.substring(
                languageName.lastIndexOf(".")+1,
           languageName.length());
43.              Constructor cons;
44.              try {
45.                   cons =
                ((Class)vPlugins.get(plugins)).getConstructor
                (Reader.class);
46.                   try {
47.                        Lexer l = (Lexer)cons.newInstance(new
                     DocumentReader(new
               HighLightedDocument(plusProcessor)));
48.                        vToken = new Vector();
49.                        vToken = l.getTokens();
50.                        vTokens.add(plugins, vToken);
51.                   } catch (InstantiationException e) {
52.
                plusProcessor.showMessage(e.getMessage(),
               "Error");
53.              } catch (IllegalAccessException e) {
54.                plusProcessor.showMessage(e.getMessage(),
                  "Error");
55.           } catch (InvocationTargetException e) {
```

```
56.                 plusProcessor.showMessage(e.getMessage(),
                  "Error");
57.             }
58.         } catch (NoSuchMethodException e) {
59.             plusProcessor.showMessage(e.getMessage(), "Error");
60.         }
61.         plusProcessor.setFontChoosers(plugins, languageName,
             vToken);
62.         plugins++;
63.         } catch (ClassNotFoundException e) {
64.         plusProcessor.showMessage(e.getMessage(), "Error");
65.         } catch (IOException e) {
66.         plusProcessor.showMessage(e.getMessage(), "Error");
67.         } catch (SecurityException e) {
68.         plusProcessor.showMessage(e.getMessage(), "Error");
69.         } catch (IllegalArgumentException e) {
70.         plusProcessor.showMessage(e.getMessage(), "Error");
71.         }
72.     }
73. }
74. public int getPlugins(){
75.         return plugins;
76. }
77. public    void    newLexer(DocumentReader    documentReader,    int
            syntaxActual) throws PluginManagerException {
78.         Lexer syntaxLexer;
79.         try {
80.             Constructor cons =
                  ((Class)vPlugins.get
             (syntaxActual)).getConstructor(Reader.class);
81.             syntaxLexer = (Lexer)
             cons.newInstance(documentReader);
82.         } catch (SecurityException e) {
83.         throw new PluginManagerException("Plugin Error",e);
84.         } catch (IllegalArgumentException e) {
85.         throw new PluginManagerException("Plugin Error",e);
86.         } catch (NoSuchMethodException e) {
87.         throw new PluginManagerException("Plugin Error",e);
88.         } catch (InstantiationException e) {
89.         throw new PluginManagerException("Plugin Error",e);
90.         } catch (IllegalAccessException e) {
91.         throw new PluginManagerException("Plugin Error",e);
92.         } catch (InvocationTargetException e) {
93.         throw new PluginManagerException("Plugin Error",e);
94.         }
95.         documentReader.setSyntaxLexer(syntaxLexer);
96.         }
97.         public Vector getVTokens(int sw){
98.         if (sw == 0)
99.             return vToken;
100.             else
101.                 return vTokens;
102.             }
103.     }
```

- PluginManager se encarga de controlar todas las acciones relacionadas con los plugins, conformando las líneas necesarias para su integración en la herramienta.

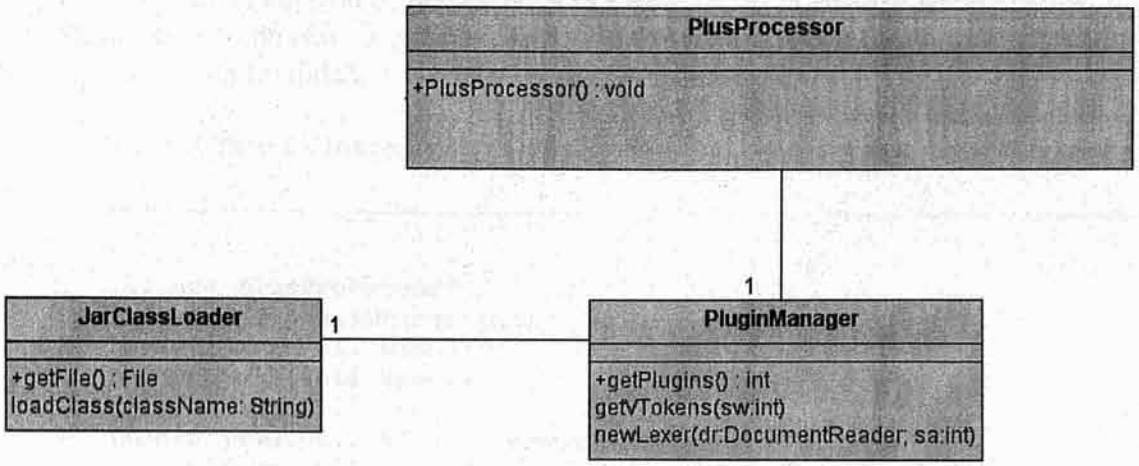
Se declararán vectores para recoger tanto las clases suministradas por los plugin como los token que aportan estas clases. (línea 14 a 15).

- En primer lugar se recogen todos los archivos .jar que cuelgan de la carpeta plugins. Cargamos el vector de plugins con una clase null, que representara al analizador léxico por defecto. De igual forma cargamos el vector de tokens con otro vector que contiene los tokens (en este caso solo contiene Default-Token) del analizador por defecto (línea 21 a 32).

- Por cada fichero .jar se busca en el archivo de manifiesto (.mf) el atributo cuyo nombre es "Language-Class-Name", se extrae la clase que coincide con el valor de ese atributo y, la clase resultante, se incluye en el vector de plugins. Pedimos el constructor de dicha clase y creamos una nueva instancia de la misma con sus correspondientes parámetros. Formamos un vector con los token correspondiente al analizador instanziado y lo guardamos el vector de tokens (línea 33 a 73).

- Esta clase posee un método para devolver el número de plugins guardados, otro para instanciar un nuevo analizador a partir de las clases guardadas en el vector de plugins y, un último método para devolver, o bien el vector de token de un determinado analizador, o bien el vector de tokens . (línea 74 a 102).

9.1.3. Diagrama de clases.



9.2. Implementación de la funcionalidad para colorear.

Se necesita de un proceso que este constantemente verificando si se produce algún cambio en el entorno de trabajo (mas si tratamos con textos que necesitan ser analizados en tiempo real). A continuación vamos a desglosar los detalles de la clase encargada de esta finalidad.

9.2.1. Clase Colorer.

```
1.  package PlusProcessor;
2.  import java.io.IOException;
3.  import java.util.HashSet;
4.  import java.util.Hashtable;
5.  import java.util.Iterator;
6.  import java.util.NoSuchElementException;
7.  import java.util.SortedSet;
8.  import java.util.TreeSet;
9.  import java.util.Vector;
10. import javax.swing.JOptionPane;
11. import javax.swing.text.SimpleAttributeSet;
12. import languageplugin.Token;
13. public class Colorer extends Thread {
14.       private DocumentReader documentReader;
15.       private HighLightedDocument document;
16.       private    TreeSet    iniPositions    =    new    TreeSet(new
            DocPositionComparator());
17.       private HashSet newPositions = new HashSet();
18.       private class RecolorEvent {
19.            public int position;
20.            public int adjustment;
21.            public RecolorEvent(int position, int adjustment) {
22.                 this.position = position;
23.                 this.adjustment = adjustment;
24.            }
25.       }
26.       private volatile Vector v = new Vector();
27.       private volatile int change = 0;
28.       private volatile int lastPosition = -1;
29.       private volatile boolean asleep = false;
30.       private Object lock = new Object();
31.       public void color(int position, int adjustment) {
32.       if (position < lastPosition) {
33.            if (lastPosition < position - adjustment) {
34.                 change -= lastPosition - position;
35.            } else {
36.                 change += adjustment;
37.            }
38.       }
39.       synchronized (lock) {
40.            v.add(new RecolorEvent(position, adjustment));
41.            if (asleep) {
42.                 this.interrupt();
43.            }
44.        }
```

```
45. }
46. public void setDocumentReader(DocumentReader dReader) {
47.       documentReader = dReader;
48. }
49. public void setDocument(HighLightedDocument document) {
50.       this.document = document;
51. }
52. public void showMessage (String message, String type){
53.       JOptionPane.showMessageDialog(null,      message,      type,
            JOptionPane.ERROR_MESSAGE);
54. }
55. public void run() {
56.       int position = -1;
57.       int adjustment = 0;
58.       boolean tryAgain = false;
59.       for (;;) { // forever
60.       synchronized (lock) {
61.       if (v.size() > 0) {
62.            RécolorEvent re = (RecolorEvent) (v.elementAt(0));
63.            v.removeElementAt(0);
64.            position = re.position;
65.            adjustment = re.adjustment;
66.           } else {
67.                tryAgain = false;
68.                position = -1;
69.                adjustment = 0;
70.           }
71.       }
72.       if (position != -1) {
73.            SortedSet workingSet;
74.            Iterator workingIt;
75.            DocPosition startRequest = new DocPosition(position);
76.            DocPosition  endRequest  =  new  DocPosition(position  +
                 ((adjustment >= 0) ? adjustment : -adjustment));
77.            DocPosition dp;
78.            DocPosition dpStart = null;
79.            DocPosition dpEnd = null;
80.            try {
81.                 workingSet = iniPositions.headSet(startRequest);
82.                 dpStart = ((DocPosition) workingSet.last());
83.                 } catch (NoSuchElementException x) {
84.                      dpStart = new DocPosition(0);
85.            }
86.            if (adjustment < 0) {
87.                 workingSet = iniPositions.subSet(startRequest,
88.                 endRequest);
89.                 workingIt = workingSet.iterator();
90.                 while (workingIt.hasNext()) {
91.                      workingIt.next();
92.                      workingIt.remove();
93.                 }
94.            }
95.            workingSet = iniPositions.tailSet(startRequest);
96.            workingIt = workingSet.iterator();
97.            while (workingIt.hasNext()) {
98.                 ((DocPosition)
                   workingIt.next()).adjustPosition(adjustment);
99.            }
100.                workingSet = iniPositions.tailSet(dpStart);
101.                workingIt = workingSet.iterator();
102.                dp = null;
```

```
103.                if (workingIt.hasNext()) {
104.                     dp = (DocPosition) workingIt.next();
105.                }
106.                try {
107.                     if (documentReader.getSyntaxActual()>0)
108.                {
109.          Token t;
110.          boolean done = false;
111.          dpEnd = dpStart;
112.          synchronized        (documentReader.getDoclock())        {
                  documentReader.getSyntaxLexer().reset(documentReader,
                        0,dpStart.getPosition(), 0);
113.              documentReader.seek(dpStart.getPosition());
114.              t = documentReader.getSyntaxLexer() .getNextToken();
115.          }
116.          newPositions.add(dpStart);
117.          while (!done && t != null) {
118.                   (documentReader.getDoclock()) {
119.              if (t.getCharEnd() <= document.getLength()) {
120.              document.setCharacterAttributes
                  (t.getCharBegin()+ change, t.getCharEnd(),
                  t.getCharBegin(),
                  (SimpleAttributeSet)
                  documentReader.getStyles().get
                  (t.getDescription()), true);
121.                    dpEnd = new DocPosition(t.getCharEnd());
122.                  }
123.               lastPosition = (t.getCharEnd() + change);
124.              }
125.              if (t.getState() == Token.INITIAL_STATE) {
126.              while (dp != null && dp.getPosition() <=
                  t.getCharEnd()) {
127.                        if (dp.getPosition() == t.getCharEnd() &&
                            dp.getPosition() >=
                  endRequest.getPosition()) {
128.                              done = true;
129.                              dp = null;
130.                        } else if (workingIt.hasNext()) {
131.                                    dp = (DocPosition) w
                                  orkingIt.next();
132.                                  } else {
133.                                       dp = null;
134.                                  }
135.                             }
136.                             newPositions.add(dpEnd);
137.                         }
138.                        synchronized (documentReader.getDoclock()) {
139.                          t =
                          documentReader
                  .getSyntaxLexer().getNextToken();
140.                        }
141.                    }
142.                workingIt = iniPositions.subSet (dpStart, dpEnd)
                    .iterator();
143.                while (workingIt.hasNext()) {
144.                     workingIt.next();
145.                     workingIt.remove();
146.                }
147.                workingIt = iniPositions.tailSet(
148.                new DocPosition(document.getLength()))
149.                .iterator();
150.
```

```
151.                while (workingIt.hasNext()) {
152.                    workingIt.next();
153.                    workingIt.remove();
154.                  }
155.                  iniPositions.addAll(newPositions);
156.                  newPositions.clear();
157.                }
158.                } catch (IOException e) {
159.                    showMessage(e.getMessage(), "Error");
160.                    }
161.                synchronized (documentReader.getDoclock()) {
162.                lastPosition = -1;
163.                change = 0;
164.                }
165.            tryAgain = true;
166.            }
167.            asleep = true;
168.            if (!tryAgain) {
169.            try {
170.                sleep(0xffffff);
171.                } catch (InterruptedException e) {
172.              }
173.            }
174.            asleep = false;
175.            }
176.        }
177.    }
```

- Para realizar el resaltado de sintaxis, se ejecuta una tarea por cada documento de texto a manipular. La clase colorer implementa esta tarea y mantiene una lista de posiciones en el fichero dónde se realiza el resaltado cada vez que el analizador reporta el estado inicial. Esta lista necesita ser almacenada (TreeSet) para posteriormente recuperarla y así tener una gestión mas eficiente del coloreado (línea 13 a 16).

- Al modificar el texto se borran y producen nuevas posiciones de texto a resaltar. Estas posiciones no pueden ser eliminadas y escritas al mismo tiempo en la lista, por tanto, se mantiene una tabla de nuevas posiciones y simplemente se añaden a la lista una vez que las posiciones invalidas hayan sido eliminadas (línea 17).

- Se necesita un envoltorio que represente el texto que necesite ser coloreado. Este envoltorio es un objeto que podrá ser almacenado a su vez en un vector (líneas 18 a 26).

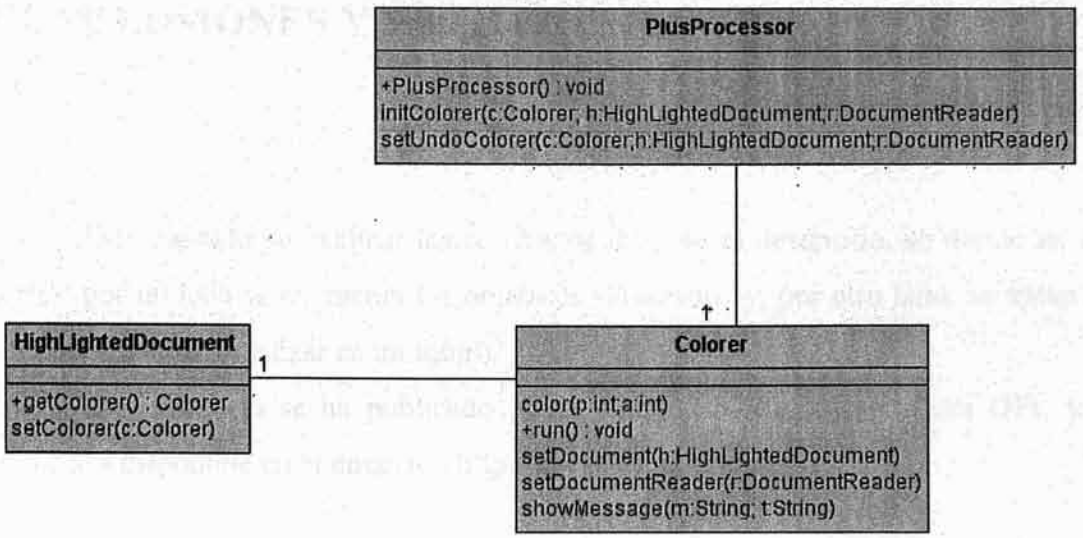
- Se deberá guardar la cantidad de cambios que han ocurrido antes del lugar en el documento que nosotros estamos coloreando (lastPosition). Además se

necesitará crear una sección crítica que dote al vector de posiciones de sincronismo cuando se realizan modificaciones en el mismo. Para ello creamos el objeto lock (líneas 27 a 30).

- Podremos pedir al coloreador tomar en cuenta una sección determinada del documento. Esto se procesará en forma de cola FIFO en un vector, por tanto, deberá de estar sincronizado. Se ajustará el lugar del documento que coge el foco de coloreado en las variables pertinentes. El coloreador se ejecuta siempre y puede estar en un estado inactivo (sleep) durante largos periodos de tiempo. Deberá ser interrumpido cada vez que haya algo que realizar. (líneas 31 a 45).

- Si se ha terminado de colorear el texto, la tarea no puede "dormir" hasta que no haya nada más que hacer. Para asegurarnos de ello, se declara la variable tryAgain (línea 58). Obtenemos la primera posición del vector de posiciones y se comprueba si es válida. De no ser así, se comenzará por el principio del documento (líneas 60 a 45). Si el ajuste es borrado, se tomará cualquier posición quitada de la lista (líneas 86 a 94). Posteriormente se ajustarán las posiciones de todo después de la inserción o borrado (líneas 95 a 99). A continuación se realiza el coloreado de posiciones potenciando al máximo la eficiencia del analizador. Se podría crear un nuevo analizador para el documento manipulado cada vez, pero es mejor reajustarlo de modo que piense que está comenzando el documento (líneas 95 a 112). Después de que se haber echo esto, se ajusta el lector del analizador de modo que esté también en el punto correcto (línea 113). Resaltamos los tokens hasta que alcanzamos el final del documento, situación devuelta por el analizador. Para ello se hace un control exhaustivo de las posiciones de los punteros de lectura y se registran las nuevas posiciones (líneas 114 a 141). Se quitan todas las viejas posiciones iniciales del lugar en donde comenzamos a hacer la el resaltado además de las posiciones que estén después del final del documento de texto (líneas 142 a 154). Se añaden en la lista las nuevas posiciones iniciales que hemos encontrado (líneas 155 y 156). Puesto que se han realizado cambios, se debe comprobar que no haya nada hacer antes de "dormir" la tarea (líneas 169 a 174).

9.2.2. Diagrama de clases.



Capítulo 10

CONCLUSIONES Y TRABAJOS FUTUROS

Este capítulo se evalúan los resultados de todo el desarrollo, se divide en dos partes: por un lado se enumeran los objetivos alcanzados y, por otro lado, se tratan los posibles trabajos a realizar en un futuro.

Este proyecto se ha publicado como código abierto bajo licencia GPL y se encuentra disponible en la dirección http://gavab.escet.urjc.es.

10.1. Conclusiones.

10.1.1. Incorporación de Plugins.

Como ha podido verse a lo largo de esta memoria, la integración de todas las herramientas de procesamiento de textos en un solo desarrollo software no es una tarea fácil. La mayor dificultad se observa cuando se intentan añadir nuevos analizadores léxicos personalizados por el usuario para uso mas adecuado para el mismo. Es razonable proporcionar una API que ofrezca una forma integrada y sencilla para facilitar la asociación de este tipo de elementos.

Una parte importante de este trabajo ha sido el desarrollo de un sistema capaz de proveer mecanismos para incorporación de plugins que permitan, por lo menos, la utilización, de nuevos analizadores lexicográficos creados de forma que cubran todas las necesidades de los usuarios.

10.1.2. Desarrollo de una API para procesamiento de textos .

Este TFC es la base que podrá utilizarse en todo tipo de herramientas que utilizan textos con resaltado de sintaxis. En especial, se puede incorporar como un editor de textos genérico, una herramienta educativa y/o de visualizaciación software, desarrollos web, etc.

PlusProcessor ha servido, además de lo anteriormente expuesto, para introducirnos en la tecnología Java, en especial en SWING.

10.2. Trabajos futuros.

Es posible completar PlusProcessor para dotarle con las características que aparecen en otros desarrollos similares. Entre ellas vamos a destacar algunas de cierto interés:

10.2.1. Code-folding.

Code-folding es una característica de ciertos procesadores de textos que permite que el usuario pliegue las secciones de un archivo del código fuente mientras que trabaja en otras partes del mismo. Esto le permite manejar cantidades más grandes de código dentro de una ventana, disminuyendo su complejidad de análisis.

10.2.2. Autocompletado de tokens.

Se puede sugerir al usuario mientras escribe la posibilidad de autocompletar los token que coinciden con la parte de texto escrita mediante una "tarea diccionario", de modo que le resulte más sencillo el desarrollo de su trabajo.

10.2.3. Plantillas.

Permitir mediante algún mecanismo (por ejemplo CTRL + Espacio) crear automáticamente una plantilla de métodos, el tipo de retorno, sus parámetros, etc. Un ejemplo en Eclipse sería el siguiente:

Bibliografía

[1] **Toub, S.** (2000). *Evaluating Information Architecture: A Practical Guide to Assessing Web Site Organization* . ARGUS Associates.

[2] **G. Kontoya e I. Sommerville.** (1997). *Requirements Engineering: Processes and Techniques.* John Wiley & Sons.

[3] **Jesus S. Allende et al.** (2005). *Java 2.* 2ª edición, McGraw-Hill.

[4] **J. Gosling et al.** (2001). *El Lenguaje de Programación Java.* 3ª edición, Addison-Wesley.

[5] **Bertrand Meyer** (2001). *Construcción de Software Orientado a Objetos.* 2ª edición Prentice Hall.

[6] **Graig Larman** (2003). *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* 2ª edición, Prentice Hall.

[7] **Newkirk J.** (2002). *La programación extrema en la práctica.* Ed. Pearson Education, Madrid.

[8] **Stephens, M y Rosenberg D.** (2003). *Programming refactored: the case against X.P.* Ed. Berkeley, California.

[9] **Buschman, Frank. et al.** (1996). *Pattern-oriented software architecture.* A system of patterns, John Wiley and Sons, West Sussex, England

[10] **Keogh, Jim.** (2003). *J2EE Manual de referencia*, McGraw-Hill, Madrid.

[11] **Blooch, Grady (1994) .** *Object-oriented analysis and design with aplications.* Second Edition, The Benjamin/Cummings Publishing Company, Inc.

[12] **James Rumbaugh (1991).** *Object-Oriented Modeling and Design.* et al. Prentice-Hall, Inc.

Internet

http://www.programacion.com/tutorial/java_basico/

http://java.sun.com/docs/books/tutorial/java/index.html

http://www.dickbaldwin.com/toc.htm

http://www.sap-img.com/java/index.htm

http://mindprod.com/jgloss/jgloss.html

http://www.elrincondelprogramador.com/default.asp?pag=articulos%2Fleer.asp&id=61

http://ostermiller.org/syntax/

http://java-source.net/open-source/parser-generators/jflex

http://www.javasoft.com/products/jdk/1.1/index.html

http://jpf.sourceforge.net/

http://www.swingall.com/

http://www.extremeprogramming.org

http://www.javahispano.org:

http://www.programacion.net/java

http://www.codeguru.com/java/Swing/index.shtml

http://www.htmlpoint.com/guidajava/java_27.htm

http://www.esus.com/docs/GetIndexPage.jsp?uid=275

http://www.desarrolloweb.com/

http://www.extremeprogramming.org/

http://www.microsoft.com/spanish/msdn/comunidad/dce/1/entrenamiento/foxpro/1.asp

http://www.abcdatos.com/tutoriales/tutorial/g23.html