# Ingeniería del Software: Laboratorio

### Cuaderno de prácticas. P4 – Mantenimiento

## CONTENIDOS

El presente cuaderno de prácticas debe servir como guía al alumno para la asimilación y estudio de los siguientes contenidos:

* Métricas de Mantenimiento del Software

## RECOMENDACIÓN DE HERRAMIENTAS

* Netbeans(http://netbeans.org/)

## EJERCICIO TUTORIZADO – Métricas de Mantenimiento del Software

### Enunciado

El objetivo de esta práctica es que el participante utilice una serie de herramientas para la obtención de diferentes métricas sobre un determinado software.

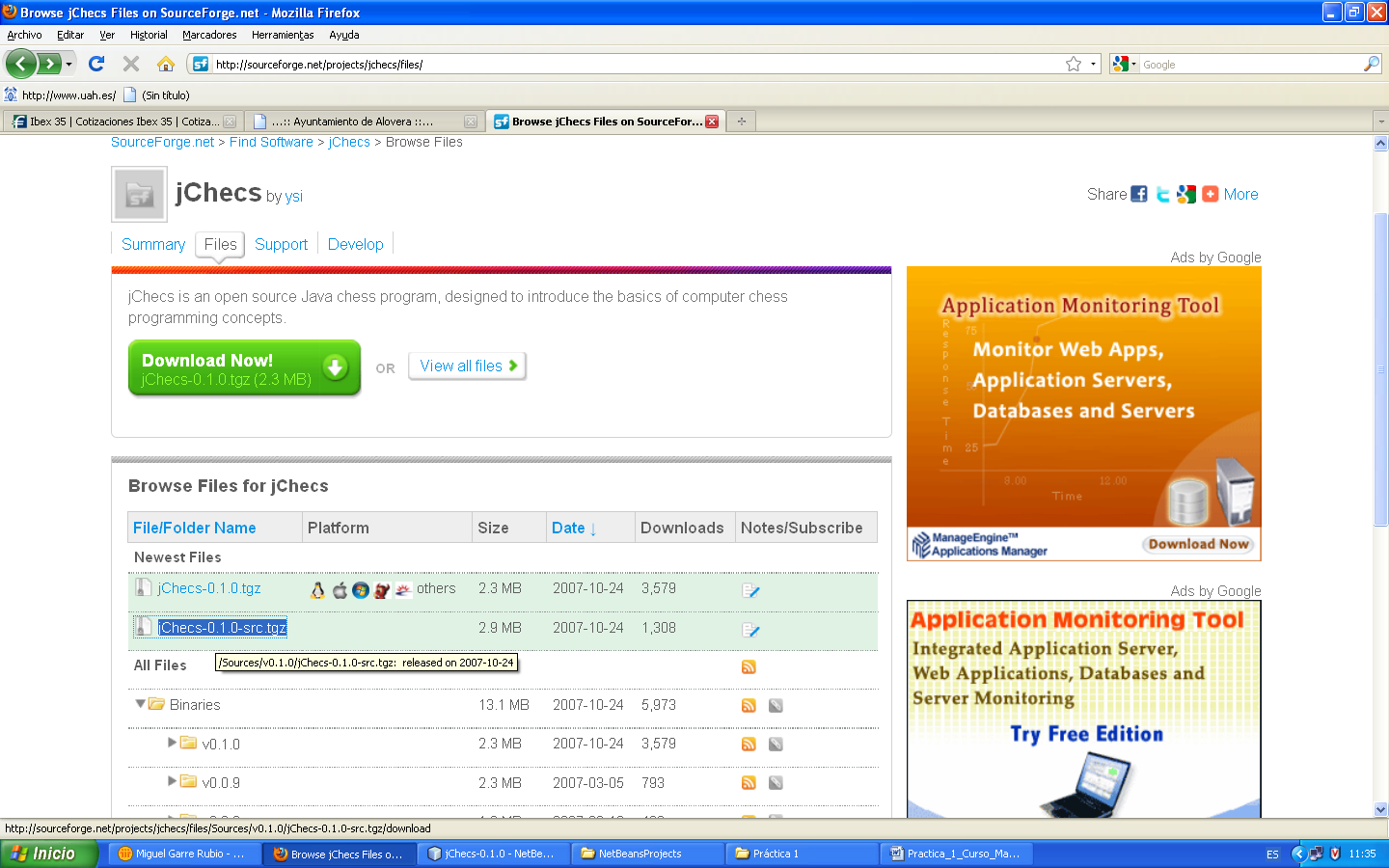
El software a medir se obtendrá del repositorio sourceforge.net, siendo un código escrito en Java con un número de clases adecuado para el propósito de esta práctica.

**Primer paso: Obtener el software sobre el que se desean realizar medidas.**

En nuestro caso el software será jChecs (programa fuente en Java, diseñado para introducir los conceptos básicos de la programación de juegos de ajedrez).

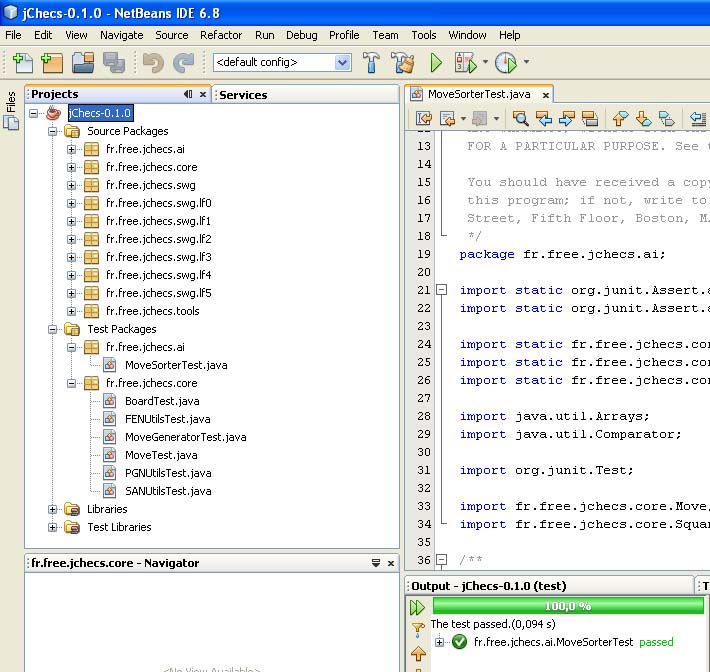
Pasos a seguir:

* 1. Abrir la página web <http://sourceforge.net/projects/jchecs/files/>*.*



Descargar el archivo [jChecs-0.1.0-src.tgz](http://sourceforge.net/projects/jchecs/files/Sources/v0.1.0/jChecs-0.1.0-src.tgz/download).

* 1. Integrar la aplicación en NetBeans.
     + Abrir NetBeans IDE 6.8 o posterior.
     + Crear un nuevo proyecto: File | New Project. En la ventana emergente **New Project** seleccionar Java como Categoría, y Java Application como Projecto.
     + En la ventana emergente **New Java Application** colocar en el campo **Project Name** el nombre *jChecs* y asegurarse de no crear la clase principal.
     + Pulsar el botón “*Finish*”. Entonces aparecerá el proyecto *jChecs* con sus carpetas asociadas, el cual puede verse en la ventana Projects.
  2. Integrar el proyecto JChecs en NetBeans:
* Descomprimir el archivo *jChecs-0.1.0-src.tgz*, en cualquier parte. Se generará el directorio *jChecs-0.1.0-src*, el cual contiene a su vez un archivo comprimido *jChecs-0.1.0-src*. Descomprimir este archivo. Se genera el directorio *jChecs-0.1.0-src*, y dentro de éste el directorio *jChecs-0.1.0*, que a su vez contiene los directorios *build* y *src*.
  + - Sustituir los directorios equivalentes que genera NetBeans, por los obtenidos anteriormente.
    - Mover las clases de test al directorio correspondiente.
    - Finalmente en NetBeans se tendrá:



**Segundo paso: Obtención de métricas mediante JavaNCSS**

**JavaNCSS** es una herramienta que brinda información sobre un código fuente Java a nivel de método, clase o paquete. Se obtiene gratuitamente de http://www.kclee.de/clemens/java/javancss/*.*

1. Después de descargar y descomprimir el archivo, la aplicación se ejecuta de la siguiente manera (desde el directorio en el cual se descomprimió javancss):

java -classpath <<librerías>> javancss.Main -gui test/\*.java

* Sustituimos <<librerías>> por la ruta de las librerías de JavaNCSS que vienen en la carpeta lib, separadas por punto y coma (por ejemplo:

.;C:\javancss-32.53\lib\javancss.jar;C:\javancss-32.53\lib\ccl.jar;C:\javancss-32.53\lib\jhbasic.jar;C:\javancss-32.53\lib\javacc.jar

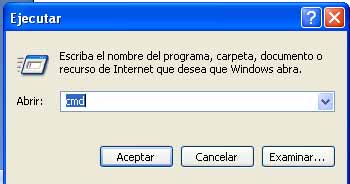
)

* El primer argumento –gui invoca la interfaz gráfica.
* El segundo argumento establece los archivos fuente para los que se realizará el análisis. En este caso los archivos ubicados en el directorio test. Si se desean analizar otros archivos habrá que indicar su ruta.
* Otra forma de ejecutar la aplicación sería, utilizando esta línea de comandos desde el directorio lib:

java -jar javancss.jar -all -recursive -gui ..\test

La opción –recursive indica que se examinarán todos los archivos .java que cuelguen del directorio indicado.

1. Obtención de las métricas de nuestro proyecto mediante JavaNCSS:
   * + Seleccionamos el menú Inicio de Windows, y dentro de éste Ejecutar.
     + Dentro de la ventana Ejecutar escribimos cmd para abrir una ventana de comandos.



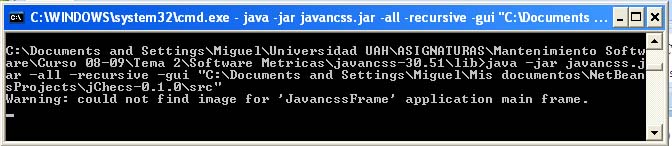
* + - Nos situamos en el directorio lib donde se encuentre javancss-32.53 (o posterior).
    - Escribimos:

java -classpath <<librerías>> javancss.Main -gui C:\jChecs-0.1.0\src

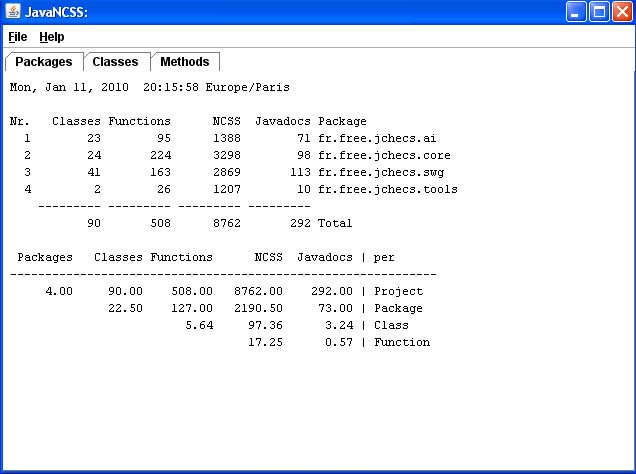
o bien …

java -jar javancss.jar -all -recursive -gui c:\*jChecs-0.1.0\*src

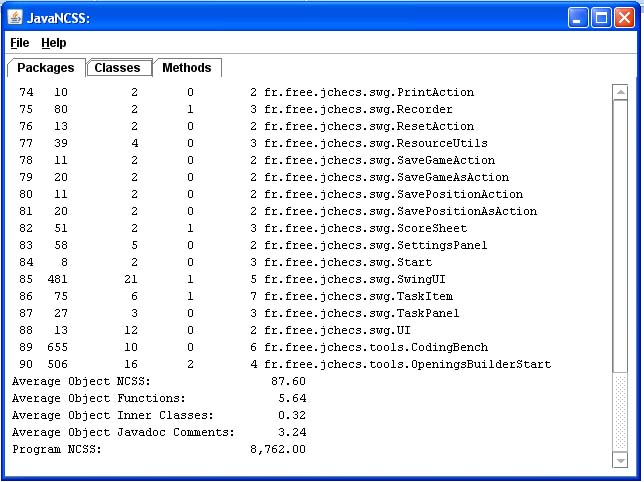
suponiendo que c:\*jChecs-0.1.0\*src es el directorio en el que se encuentran los archivos fuente del proyecto. Será necesario utilizar comillas si existen nombres de directorios con espacios.



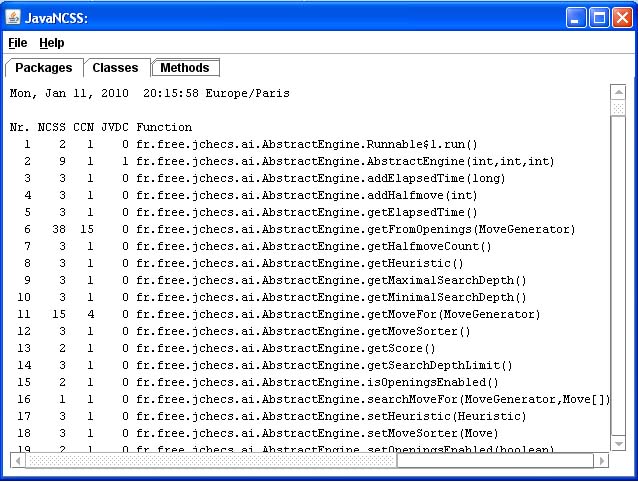
* + - La salida se obtiene a través de una interface gráfica:



Destacar el significado de NCSS – *Non commenting source statements.* Las funciones son los métodos de las clases.



Las clases fr.free.jchecs.core.MailboxBoard y fr.free.jchecs.core.X88Board son las que más métodos tienen (25).



A nivel de método encontramos dos métricas adicionales, CCN número de complejidad ciclomática y JVDC comentarios Javadocs.

Algunas de las clases con mayor complejidad ciclomática son:

fr.free.jchecs.core.ArrayBoard.isAttacked (CCN 92)

fr.free.jchecs.core.MailboxBoard.isAttacked (CCN 82)

fr.free.jchecs.core.X88Board.isAttacked (CCN 82)

fr.free.jchecs.ai.MobilityHeuristic.evaluate (CCN 47

fr.free.jchecs.tools.CodingBench.benchIterations (CCN 44)

fr.free.jchecs.core.PGNUtils.toGame (CCN 42)

Siendo estas dos últimas, las clases con más métodos.

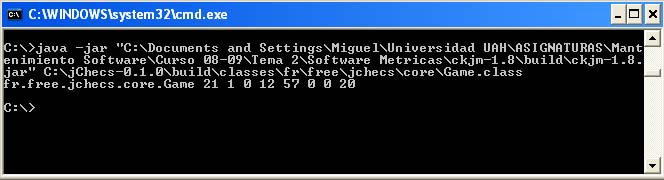
Esta información se puede grabar File | Save, generándose tres archivos en el directorio javancss-32.53: javancss.packages.txt, javancss.classes.txt y javancss.methods.txt, los cuales contienen de forma respectiva la información sobre paquetes, clases y métodos.

**Tercer paso: Obtención de métricas mediante CKJM**

**CKJM** es un programa que calcula las métricas en orientación a objetos Chidamber & Kemerer, procesando el bytecode de un fichero java compilado, es decir, los archivos .class.

1. CKMJ puede descargarse gratuitamente de: <http://www.spinellis.gr/sw/ckjm/>. Tras descargar y descomprimir el fichero invocamos el programa ejecutando java con la opción –jar pasándole como argumento la localización del archivo ckjm.jar que obtuvimos anteriormente. Además especificamos el fichero .class que queremos analizar, en el siguiente ejemplo seria Game.class.

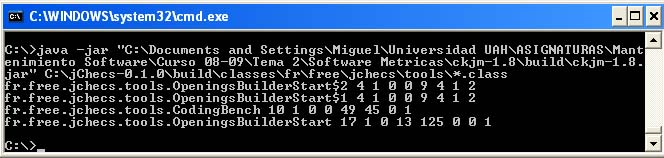
C:\java –jar C:\ckjm-1.9\build\ckjm-1.9.jar C:\jChecs-0.1.0\build\classes\fr\free\jchecs\core\Game.class



Los datos obtenidos son:

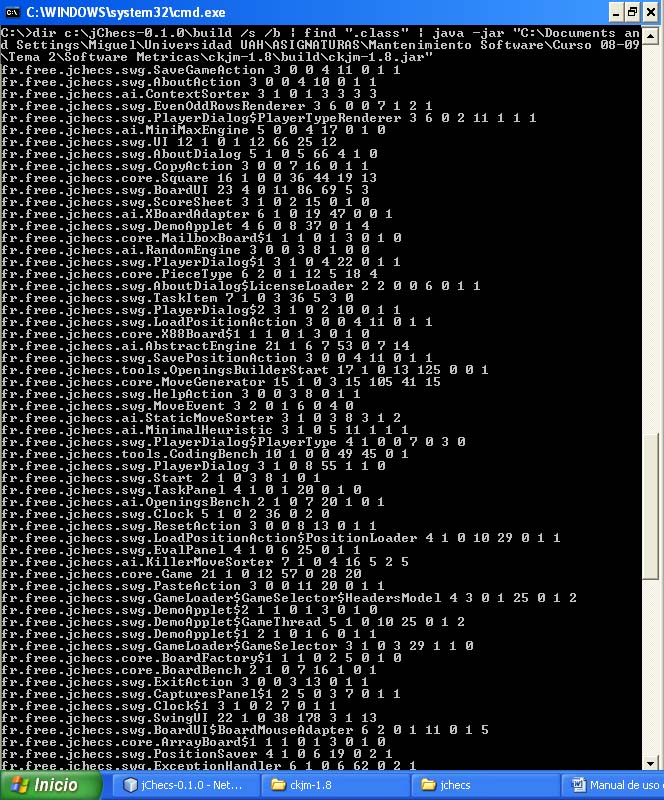
* Weighted methods per class (WMC)
* Depth of Inheritance Tree (DIT)
* Number of Children (NOC)
* Coupling between object classes (CBO)
* Response for a Class (RFC)
* Lack of cohesion in methods (LCOM)
* Afferent coupling (not a C&K metric) (Ca)
* Number of Public Methods for a class (not a C&K metric) (NPM)
  1. Para analizar todas las clases de un paquete especificamos la carpeta que las contiene y dentro de ella todos los archivos con extensión .class, así vemos las métricas del paquete fr.free.jchecs.tools.

C:\java –jar C:\ckjm-1.9\build\ckjm-1.9.jar C:\jChecs-0.1.0\build\classes\fr\free\jchecs\tools\\*.class



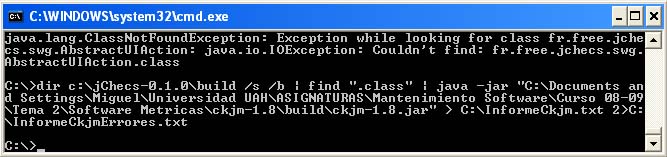
* 1. Si se quieren obtener las métricas para todas las clases de un proyecto, habría que obtener todos los archivos del directorio build y de sus subdirectorios con el comando dir y la opción /s. A continuación se filtrarían los archivos que posean la extensión .class. Esa salida se la pasaríamos a la herramienta ckjm como se muestra a continuación.

C:\>dir c:\jChecs-0.1.0\build /s /b | find “.class” | java –jar C:\ckjm-1.9\build\ckjm-1.9.jar



* 1. Para guardar el resultado obtenido, se debe redireccionar la salida obtenida hacia un fichero de texto, así como la salida con los mensajes de error.

C:\>dir c:\jChecs-0.1.0\build /s /b | find “.class” | java –jar C:\ckjm-1.9\build\ckjm-1.9.jar > C:\InformeCkjm.txt 2> C:\InformeCkjmErrores.txt



Veamos a continuación un resumen de las clases con peor mantenibilidad.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | (WMC) | (DIT) | (NOC) | (CBO) | (RFC) | (LCOM) |
| fr.free.jchecs.ai.AbstractEngine | 21 | 1 | 6 | 7 | 53 | 0 |
| fr.free.jchecs.ai.Engine | 14 | 1 | 0 | 3 | 14 | 91 |
| fr.free.jchecs.ai.XBoardAdapter | 6 | 1 | 0 | 19 | 47 | 0 |
| fr.free.jchecs.core.AbstractBoard | 23 | 1 | 2 | 3 | 51 | 65 |
| fr.free.jchecs.core.ArrayBoard | 21 | 0 | 0 | 8 | 65 | 0 |
| fr.free.jchecs.core.FENUtils | 11 | 1 | 0 | 5 | 47 | 13 |
| fr.free.jchecs.core.Game | 21 | 1 | 0 | 12 | 57 | 0 |
| fr.free.jchecs.core.MailboxBoard | 26 | 0 | 0 | 8 | 69 | 0 |
| fr.free.jchecs.core.MoveGenerator | 15 | 1 | 0 | 3 | 15 | 105 |
| fr.free.jchecs.core.MoveGeneratorBench | 13 | 1 | 0 | 7 | 38 | 78 |
| fr.free.jchecs.core.PGNUtils | 6 | 1 | 0 | 17 | 59 | 0 |
| fr.free.jchecs.core.SANUtils | 4 | 1 | 0 | 6 | 50 | 0 |
| fr.free.jchecs.core.Square | 16 | 1 | 0 | 0 | 36 | 44 |
| fr.free.jchecs.core.X88Board | 26 | 0 | 0 | 8 | 69 | 0 |
| fr.free.jchecs.swg.AboutDialog | 5 | 1 | 0 | 5 | 66 | 4 |
| fr.free.jchecs.swg.AbstractUIAction | 4 | 2 | 13 | 2 | 16 | 0 |
| fr.free.jchecs.swg.BoardUI | 23 | 4 | 0 | 11 | 86 | 69 |
| fr.free.jchecs.swg.DemoApplet$GameThread | 5 | 1 | 0 | 10 | 25 | 0 |
| fr.free.jchecs.swg.EvenOddRowsRenderer | 3 | 6 | 0 | 0 | 7 | 1 |
| fr.free.jchecs.swg.ExceptionHandler | 6 | 1 | 0 | 6 | 62 | 0 |
| fr.free.jchecs.swg.GameLoader | 4 | 1 | 0 | 10 | 35 | 0 |
| fr.free.jchecs.swg.LoadPositionAction$PositionLoader | 4 | 1 | 0 | 10 | 29 | 0 |
| fr.free.jchecs.swg.PasteAction | 3 | 0 | 0 | 11 | 20 | 0 |
| fr.free.jchecs.swg.PlayerDialog | 3 | 1 | 0 | 8 | 55 | 1 |
| fr.free.jchecs.swg.PlayerDialog$PlayerTypeRenderer | 3 | 6 | 0 | 2 | 11 | 1 |
| fr.free.jchecs.swg.SwingUI | 22 | 1 | 0 | 38 | 178 | 3 |
| fr.free.jchecs.swg.UI | 12 | 1 | 0 | 1 | 12 | 66 |
| fr.free.jchecs.tools.CodingBench | 10 | 1 | 0 | 0 | 49 | 45 |
| fr.free.jchecs.tools.OpeningsBuilderStart | 17 | 1 | 0 | 13 | 125 | 0 |

En rojo se han marcado las clases que con mayor claridad son más problemáticas desde el punto de vista del mantenimiento, y en naranja las que a simple vista también muestran problemas, pero que habría que estudiar en mayor detenimiento.

**Cuarto paso: Obtención de métricas mediante Dependency Finder.**

Dependency Finder es una suit de herramientas para analizar cualquier código Java compilado (.jar, .zip, .class). Analizando las características internas del código, es capaz de obtener grafos de dependencias, así como otras medidas de utilidad. Puede descargarse gratuitamente de <http://depfind.sourceforge.net/>.

Una dependencia se produce cuando una clase usa servicios de otra, por ejemplo posee un atributo de otra clase o sus métodos llaman a métodos ajenos. La dependencia se manifiesta cuando un elemento A requiere la presencia de otro B, entonces decimos que A depende de B y lo representamos por A→B.

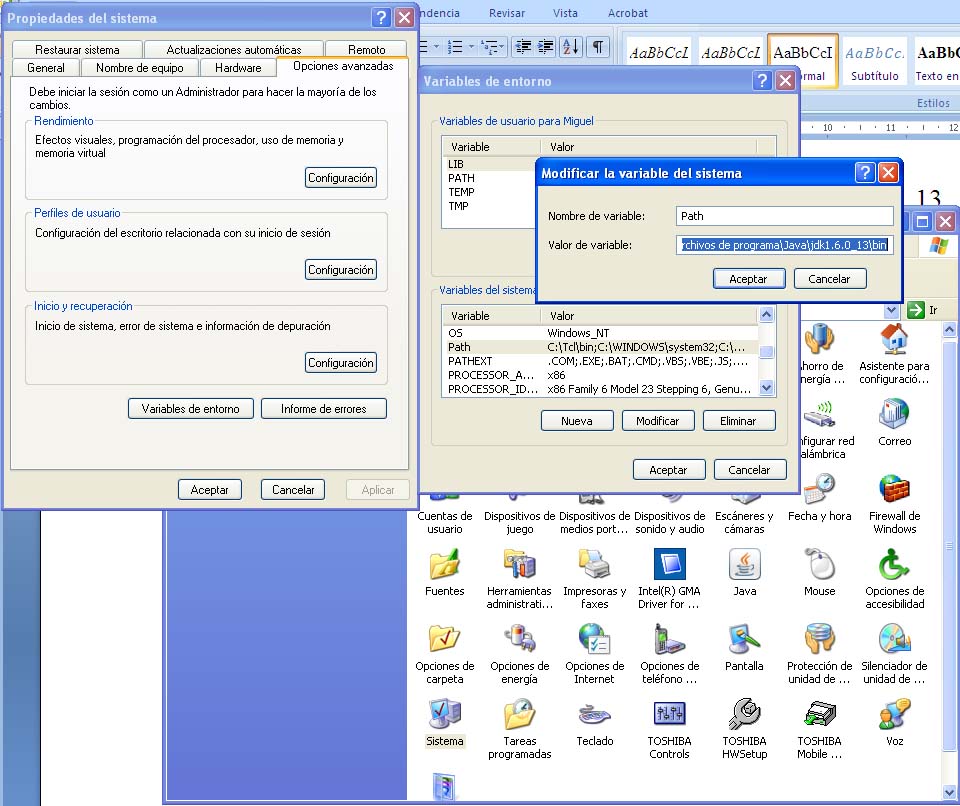
1. Después de descargar y descomprimir el fichero, comprobamos que la variable de entorno JAVA\_HOME posee la dirección del JDK instalado, por ejemplo: C:\Archivos de programa\Java\jdk1.6.0\_13\bin. Si esta variable no existe habrá que crearla. Además hay que incluir en la variable PATH la dirección a la carpeta bin dentro de DependencyFinder-1.2.1-beta4:

JAVA\_HOME= C:\Archivos de programa\Java\jdk1.6.0\_13

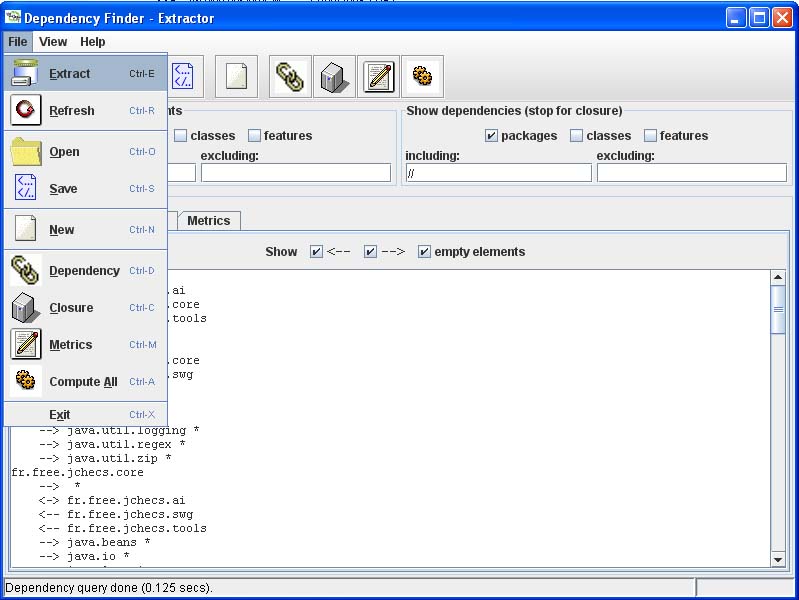
PATH= C:\DependencyFinder-1.2.1-beta4\bin;C:\Archivos de programa\Java\jdk1.6.0\_13\bin

Para hacer esto hay que:

* Pulsar en el menú Inicio de Windows Inicio | Panel de Control. Se abre la ventana Panel de Control, y dentro de ésta seleccionamos Sistema.
* En la ventana Propiedades del Sistema pulsamos sobre la pestaña Opciones Avanzadas, y dentro de ésta pulsaremos sobre el botón Variables de Entorno.
* La ventana Variables de Entorno permite ver las variables de usuario y de sistema existentes, así como sus valores. Podemos modificar, eliminar o añadir variables. En nuestro caso añadiremos la variable JAVA\_HOME, y modificaremos la variable PATH, todo ello desde la parte Variables del sistema.



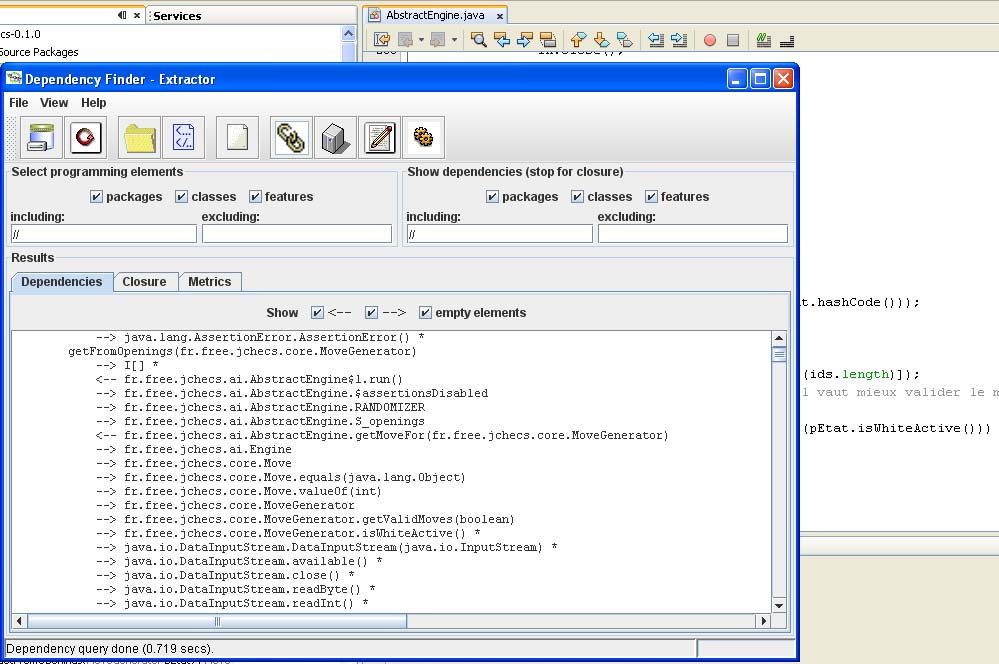
1. Por último ejecutamos DependencyFinder.bat que se encuentra en el directorio bin de la herramienta. Para seleccionar un proyecto utilizamos File | Extract. Para obtener el gráfico de dependencias File | Dependency.



Se utiliza \* para distinguir identificadores que han sido referenciados en el código analizado, pero que no se estudiarán ya que generalmente son elementos de alguna librería java.

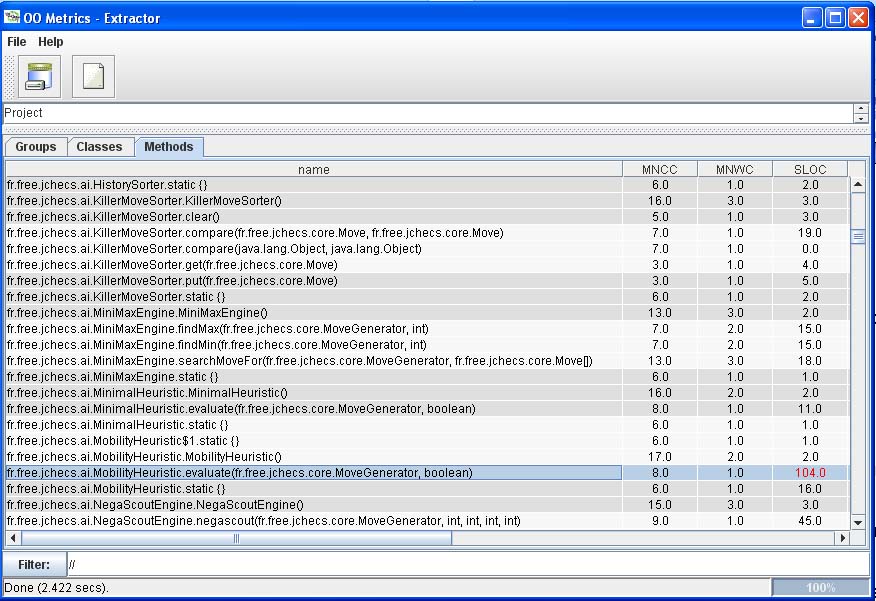
El informe mostrado está compuesto por nodos y links. Los nodos representan la estructura de nuestro proyecto (paquetes, clases, métodos), los links son las dependencias. Podemos filtrar nuestra búsqueda utilizando las secciones “*Select programming elements”* y *“Show dependencies”*. En ellas encontramos *checkboxes* para buscar sólo paquetes, clases, métodos, variables (*features* se refiere a las características internas de las clases) o combinaciones de ellos.

1. Si marcamos todos los *checkboxes* obtendríamos información detallada, apareciendo el uso que dentro de una clase se hace de los métodos de otras clases, por ejemplo:



En esta imagen se puede observar que la clase AbstractEngine tiene una dependencia con la clase MoveGenerator del paquete fr.free.jchecs.core. Más aún, utiliza uno de sus métodos, isWhiteActive. En el código se puede observar la llamada a este método a través de un objeto pEtat de la clase (pEtat.isWhiteActive()).

1. Otra herramienta muy interesante es OOMetricsGUI (situada también en el directorio bin), la cual permite obtener una serie de medidas a nivel de paquete, clase y de método. El archivo a ejecutar es OOMetricsGUI.bat.



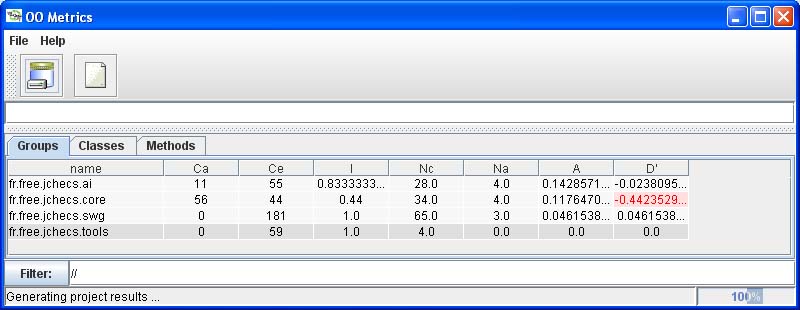
Se obtienen bastantes métricas, como por ejemplo el número de líneas de código de una clase, de un método; el número de métodos de una clase, el número de clases de un paquete, etc. En la imagen anterior se puede apreciar en rojo el número de líneas de código (SLOC) del método evaluate de la clase MobilityHeuristic, del paquete fr.free.jchecs.ai, indicando de esta forma un valor demasiado alto.

1. También es posible calcular el Índice de Inestabilidad de Robert Martin, ejecutando OOMetricsGUI.bat con el argumento –configuration y la dirección del fichero MetricsConfig.xml, que se encuentra en la carpeta etc:

OOMetricsGUI.bat -configuration C:\DependencyFinder-1.2.4-beta\etc\MartinConfig.xml

o

OOMetricsGUI.bat -configuration ..\etc\MartinConfig.xml



*Ca: afferent coupling. The number of classes outside this package that depend on classes inside this package (i.e., inbound dependencies)*

*Ce: efferent coupling. Number of classes outside this package that classes in the package depend on (i.e., outbound dependencies)*

*I: Inestability. I = Ce / (Ca + Ce)*

*A: Abstractness. A = Na / Nc*

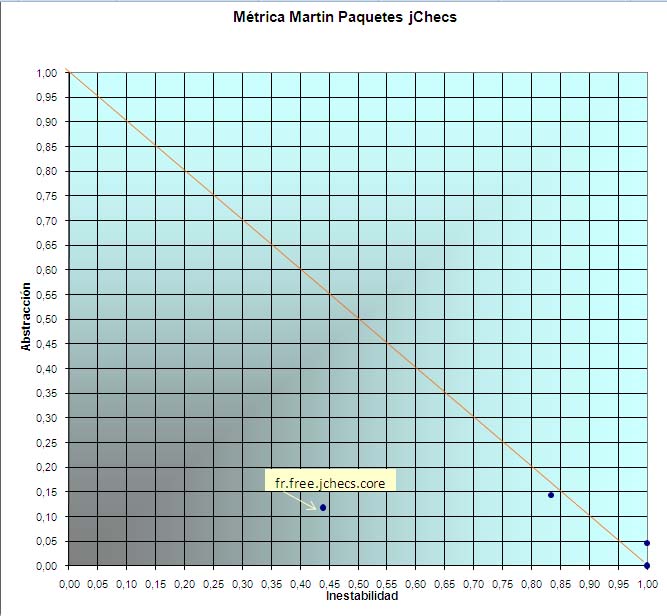
*Nc: Number of Classes*

*Na: Number of Abstract Classes and Interfaces*

*D’: Normalized Distance. D' = A + I - 1*

Se puede observar que el paquete fr.free.jchecs.core tiene un número elevado para Ca, de manera que si se realizan cambios en el mismo, éste afectará a numerosas clases, las cuales (muy probablemente) a su vez tendrán que sufrir alguna modificación.

La Inestabilidad puede tomar valores en el rango comprendido entre 0 y 1. Valores cercanos a 1 indican máxima estabilidad, pero poca flexibilidad para modificar el software. Sin embargo valores muy bajos indican demasiada volatilidad. Sin embargo, para extraer conclusiones más certeras se suelen tener en cuenta tanto la Inestabilidad como la Abstracción, de manera que no es malo tener valores grandes de I si se tienen valores pequeños de A y no es malo tener valores pequeños de I si se tienen valores grandes de A. En otras palabras, lo mejor es un valor D’ cercano al cero. Veamos gráficamente la relación entre I y A, así como los valores para los paquetes de nuestro ejemplo:



En esta figura se observa como el paquete fr.free.jchecs.core se aleja de la *Main Sequence*, tal y como se esperaba con los datos obtenidos para D’. Este debería ser un paquete a analizar en mayor detalle.

## EJERCICIO Propuesto – Métricas de Mantenimiento del Software

Ejecutando el ejercicio tutelado anterior, examinar detalladamente los resultados (métricas) obtenidos con cada una de las herramientas, tratando de establecer relaciones entre ellos, extrayendo unas conclusiones finales en referencia a la mantenibilidad de *jChecs.*