



Eusko Jaurlaritzaren Informatika Elkarte
Sociedad Informática del Gobierno Vasco

Checkstyle:

Manual de usuario

Fecha: 15/12/2014

Referencia:

EJIE S.A.
Mediterráneo, 3
Tel. 945 01 73 00*
Fax. 945 01 73 01
01010 Vitoria-Gasteiz
Posta-kutxatila / Apartado: 809
01080 Vitoria-Gasteiz
www.ejie.es

Control de documentación

Título de documento: Checkstyle

Histórico de versiones

Código:

Versión: 1.2

Fecha: 15/12/2014

Resumen de cambios: Se revisa para la versión 5.3 del eclipse Plugin. Se quita la parte de GUI, fuera del eclipse.

Versión: 1.1

Fecha: 11/05/2009

Resumen de cambios: Sólo un fichero de reglas, checkconfig.xml, en lugar de dos.

Versión: 1.0

Fecha: 07/09/2007

Resumen de cambios:

Cambios producidos desde la última versión

Segunda versión.

Control de difusión

Responsable: [Redacted]

Aprobado por: [Redacted]

Firma:

Fecha:

Distribución:

Referencias de archivo

Autor: Consultoría de áreas de conocimiento

Nombre archivo: Checkstyle. Manual de usuario v1.2.doc

Localización:

| Contenido | | |
|-----------|---|--------|
| | Capítulo/sección | Página |
| 1 | Introducción | 5 |
| 2 | Conceptos básicos | 5 |
| 3 | Integración con Eclipse | 5 |
| 3.1 | Configuración básica | 5 |
| 3.2 | Parametrización personalizada | 7 |
| 3.3 | Ejecución de Checkstyle | 10 |
| 4 | Utilidad práctica | 13 |
| 4.1 | Aportaciones de Checkstyle | 13 |
| 4.2 | Aportaciones personalizadas | 14 |
| 4.3 | Creación de reglas de validación personalizadas | 14 |
| 4.3.1. | Configuración de las reglas para el plug-in Checkstyle de Eclipse | 16 |
| 5 | Tareas ant en servidor | 19 |
| 6 | Anexo 1: Ejemplo | 20 |
| 6.1 | Resolución | 20 |

1 Introducción

En este manual se describen los distintos aspectos que debe conocer el usuario sobre Checkstyle y sobre su integración con Eclipse, así como los pasos a seguir para la creación y uso de reglas personalizadas.

2 Conceptos básicos

Checkstyle es una herramienta de desarrollo que ayuda a los programadores a escribir código Java adscrito a estándares de codificación establecidos, facilitando para ello la automatización del proceso de chequeo del código generado.

Checkstyle, por defecto, incorpora las recomendaciones de Sun sobre el estilo de código, pero estas reglas pueden ser redefinidas e incluso creadas completamente desde cero por el usuario, lo que convierte este plug-in en adaptable al estilo de codificación interno de nuestro entorno, sea cual sea.

Checkstyle puede instalarse en servidor para ser ejecutado desde tareas ant, o como plug-in de Eclipse en PC local.

Para obtener información adicional sobre el producto acceder a su página web:

<http://checkstyle.sourceforge.net>

O al respecto de su plug-in para Eclipse:

<http://eclipse-cs.sourceforge.net>

El presente documento cubre entonces el uso de Checkstyle en PC cliente como plug-in de Eclipse y su ejecución como tareas “ant”.

3 Integración con Eclipse

Al igual que en otros muchos casos, la integración del Checkstyle (como plug-in) con eclipse (o cualquier otro IDE de desarrollo) da como resultado un entorno de trabajo visual, sencillo de usar e intuitivo. Gracias a este entorno, el manejo de los aspectos de configuración y las distintas funcionalidades del Checkstyle se realiza de forma rápida y eficaz.

3.1 Configuración básica

El conjunto de reglas que utiliza Checkstyle para comprobar el código no son fijas, sino que pueden configurarse. El conjunto de reglas (criterios) que se aplican sobre un determinado proyecto vienen dadas por un fichero de configuración en formato xml, dentro de estos ficheros se especifica qué reglas serán las que inspeccionen el código, los parámetros de configuración para cada regla y el nivel de severidad del mensaje que se muestra cuando se encuentra un problema. En el caso de EJIE, existen un ficheros de configuración (**checkconfig.xml**) preparados para cubrir las exigencias de EJIE.

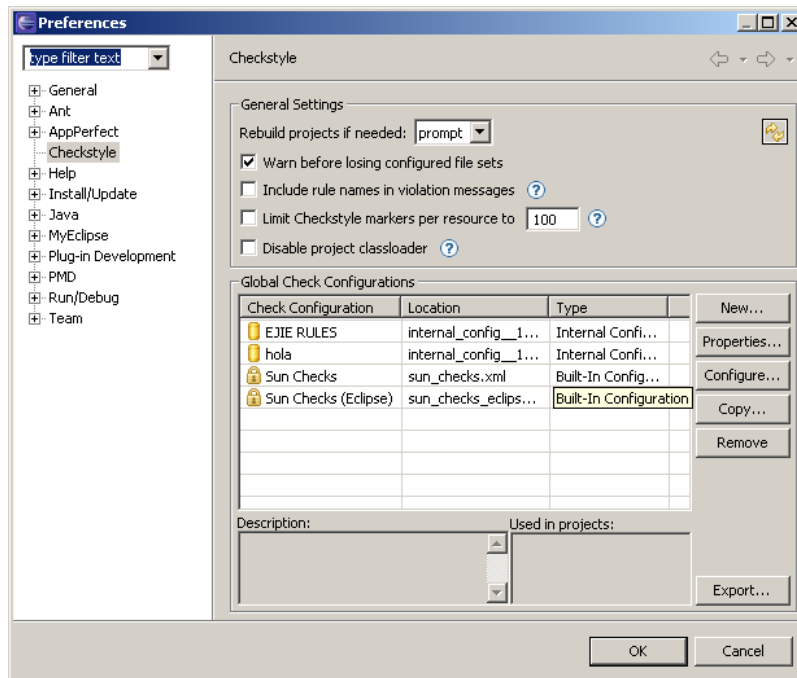
Para configurar Checkstyle con la configuración determinada para EJIE se debe seguir el siguiente proceso:

1. Descargar el fichero de reglas de Checkstyle ubicado en el SPS de OTC en:

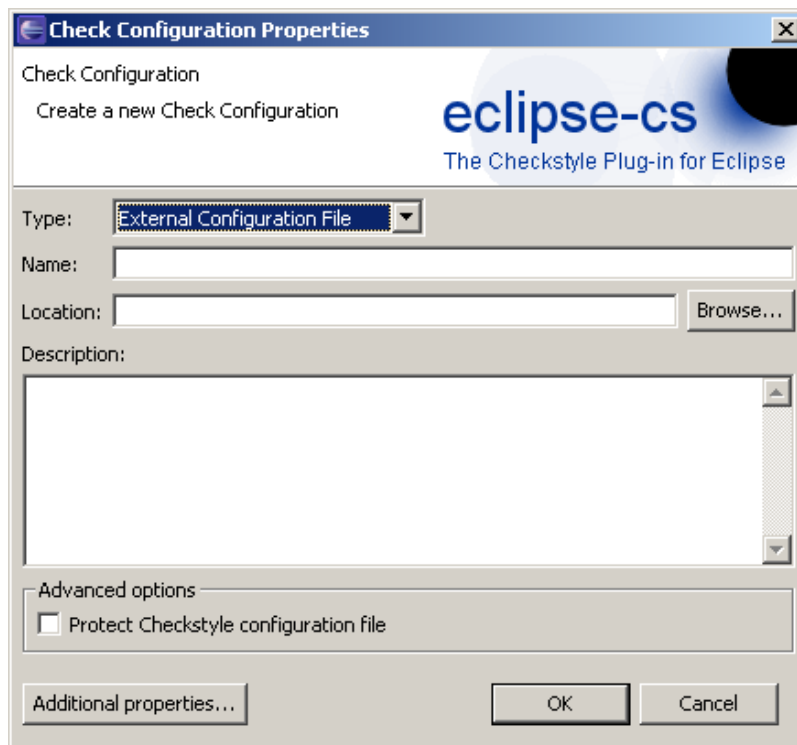
[http://elkarlan.ejje/webguneak/otc/Entregables/Repositorio%20Base/HERRAMIENTAS/Herramientas%20de%20Desarrollo%20\(PC\)/Plugins%20Analisis%20código%20estático/Calidad%202.0/reglas%20Checkstyle/OTC_Sonar_Checkconfig_5.3.xml](http://elkarlan.ejje/webguneak/otc/Entregables/Repositorio%20Base/HERRAMIENTAS/Herramientas%20de%20Desarrollo%20(PC)/Plugins%20Analisis%20código%20estático/Calidad%202.0/reglas%20Checkstyle/OTC_Sonar_Checkconfig_5.3.xml)

Se trata de un fichero checkconfig.xml. Guardar el fichero en un directorio en el puesto.

2. Arrancar el Eclipse.
3. Una vez arrancado el Eclipse, se accede al menú “window->preferences”.
4. Se selecciona en el árbol de la izquierda la sección de Checkstyle.



5. Se presiona el botón “new”.



Con “Type External Configuration File” seleccionado se cumplimentan los datos necesarios

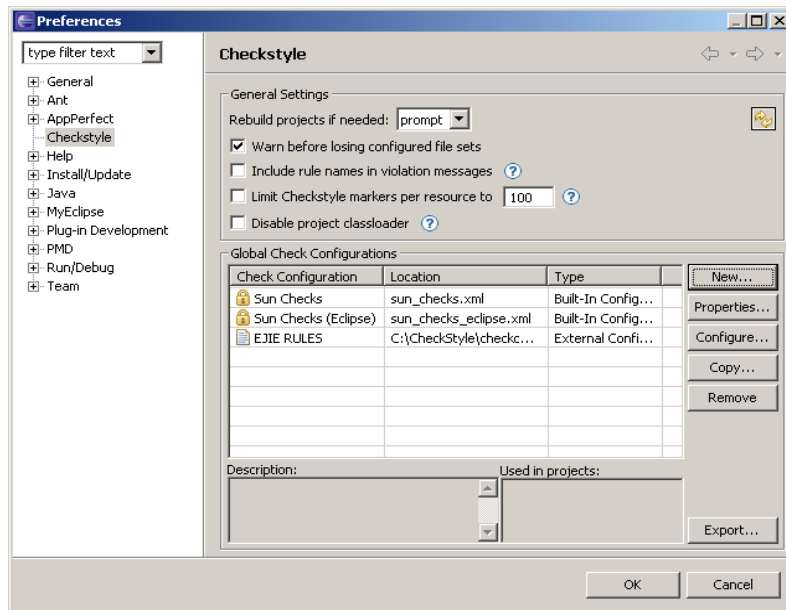
- Name→ EJIE Checkstyle Project
- Location→ La ruta donde hayamos descargado el fichero checkconfig.xml

6. Pulsar OK.

7. Una vez se completa el proceso, la aplicación podrá ser configurada para usar el fichero de reglas correspondiente a las especificaciones de EJIE.

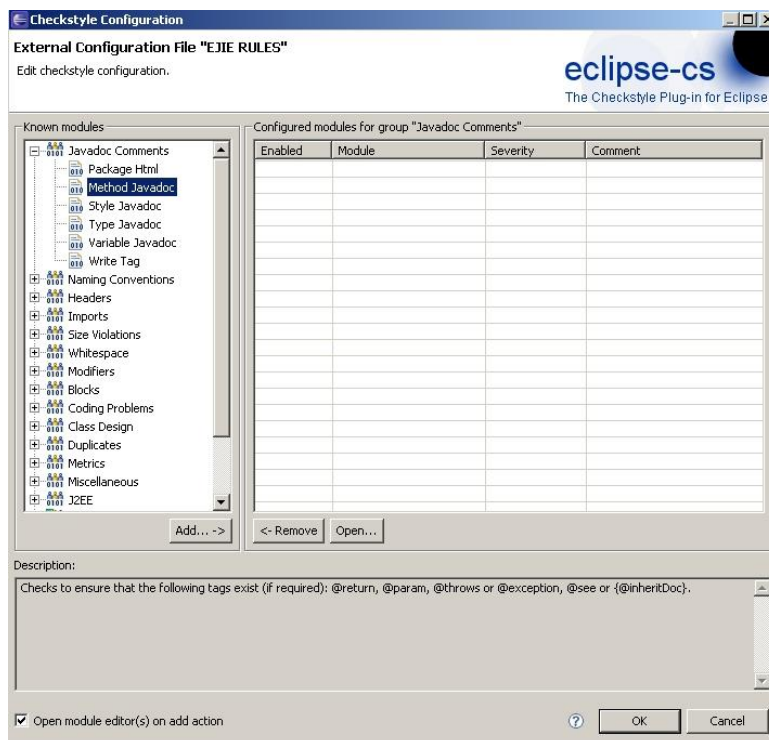
3.2 Parametrización personalizada

Además de las configuraciones propias de la aplicación y las añadidas por EJIE, también se pueden crear nuevas configuraciones o se pueden alterar configuraciones ya creadas para adaptar o mejorar las comprobaciones efectuadas por el Checkstyle. La gestión de configuraciones del Checkstyle se realiza mediante el apartado de preferencias; su acceso se realiza mediante el menú “Window-> Preferences” sección de Checkstyle.



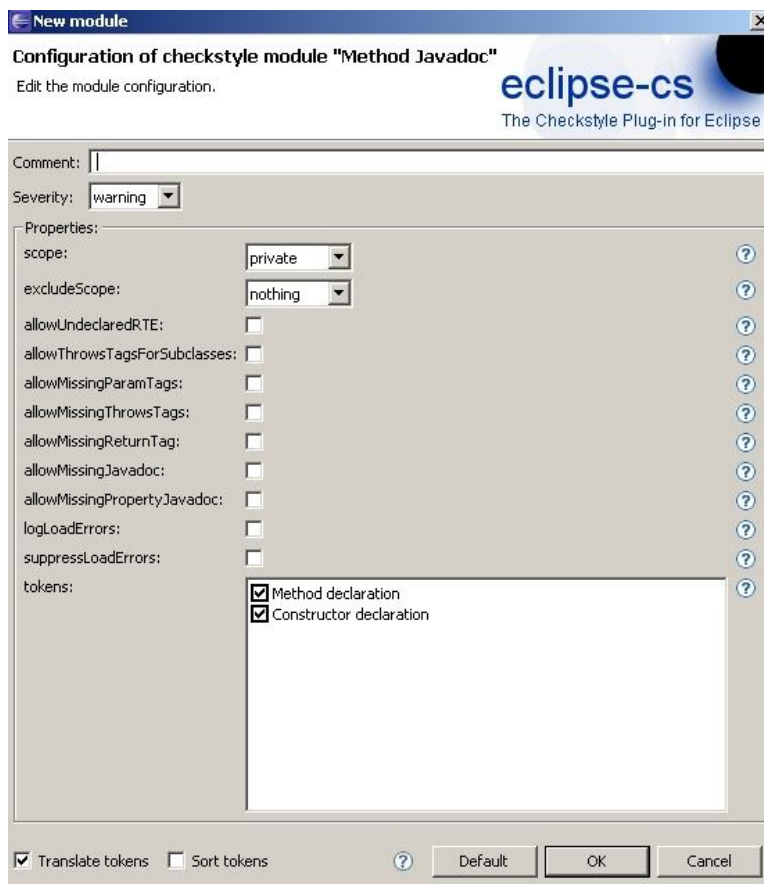
En esta ventana aparecen todas las opciones relativas a la gestión de las configuraciones (crear, editar, configurar, copiar y borrar). En esta ventana además, se pueden configurar algunas opciones generales del plug-in.

Si se deseara editar una configuración para modificarla, bastaría con seleccionar la configuración que se desee editar y presionar el botón “configure”.

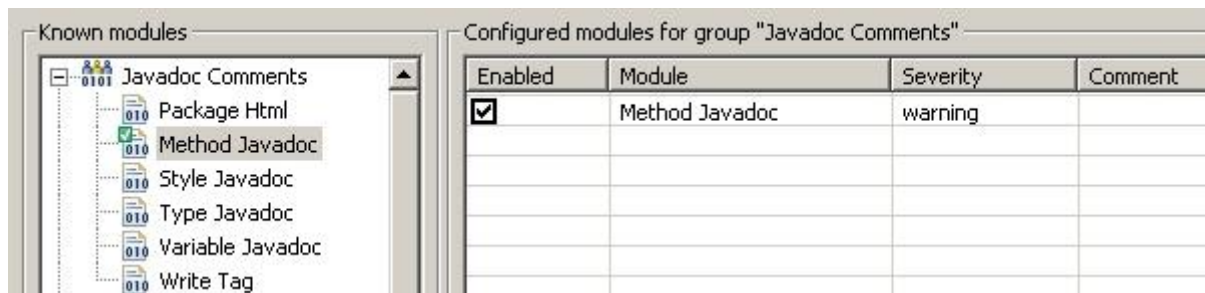


En la ventana que emergerá al apretar el botón “configure”, aparecerán (árbol de la parte izquierda) todas las opciones configurables para el testeo de código que integran el Checkstyle agrupadas por tipos. El abanico de distintas opciones de las que se dispone es bastante amplio; si se deseara mas información sobre las mismas o sobre las opciones de configuración generales en la página Web oficial de la aplicación (<http://checkstyle.sourceforge.net>) se describen todos los aspectos referentes a estas opciones de forma detallada.

Como un ejemplo sencillo, si se desea añadir a la configuración las comprobaciones del módulo “Method Javadoc”, perteneciente al grupo “Javadoc Comments”, se seleccionará dicho módulo y se pulsará sobre el botón *Add... ->*.

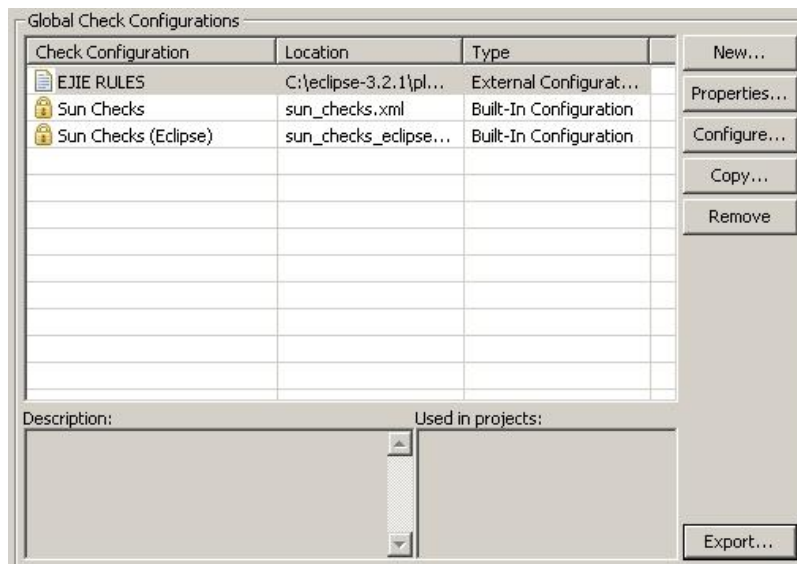


En esta nueva ventana (*New Module*) se podrán seleccionar las comprobaciones a realizar para dicho módulo “Method Javadoc”. Pulsando sobre el botón *OK* se volverá a la ventana anterior y podrá observarse que el módulo aparece ahora habilitado.



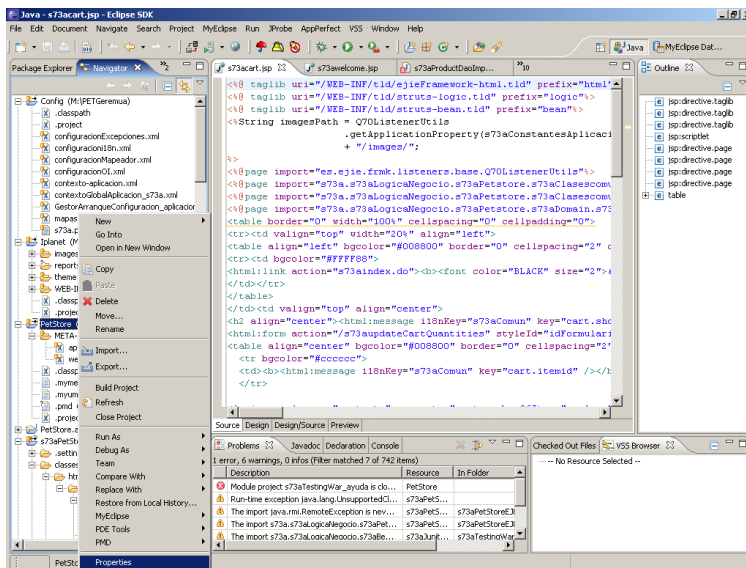
Existe la posibilidad de habilitar o deshabilitar estos módulos activando o desactivando las casillas de verificación asociadas a cada uno de ellos.

Finalmente comentar la posibilidad de exportar la configuración modificada haciendo uso del botón *Export...* situado en la ventana de preferencias de Checkstyle.

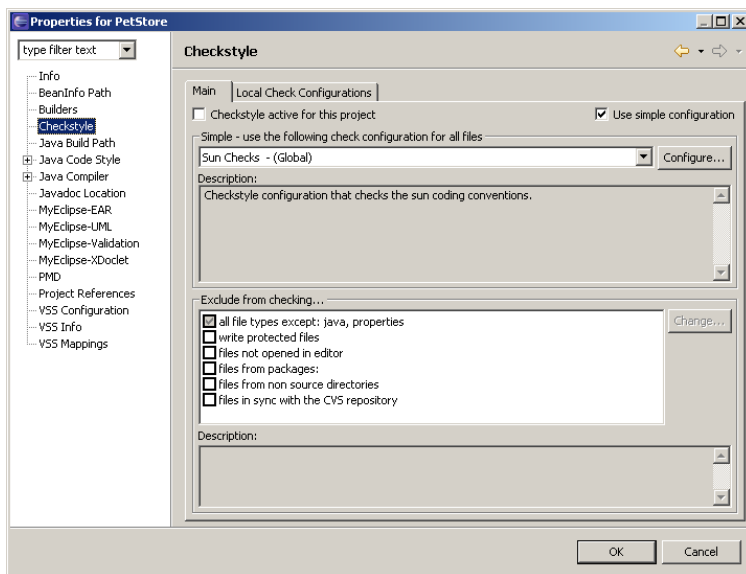


3.3 Ejecución de Checkstyle

Checkstyle, al estar integrado con Eclipse se arranca al lanzar Eclipse. El proceso de arranque, es tan simple como lanzar el ejecutable del eclipse donde se tenga configurado el plug-in del Checkstyle. Para comprobar el correcto funcionamiento de Checkstyle con Eclipse, una vez arrancado este debe aparecer la opción correspondiente del Checkstyle dentro de la ventana "properties" de cualquier proyecto. La forma más sencilla de acceder a esta ventana es, pinchando con el botón derecho sobre un proyecto y seleccionar la opción "properties".

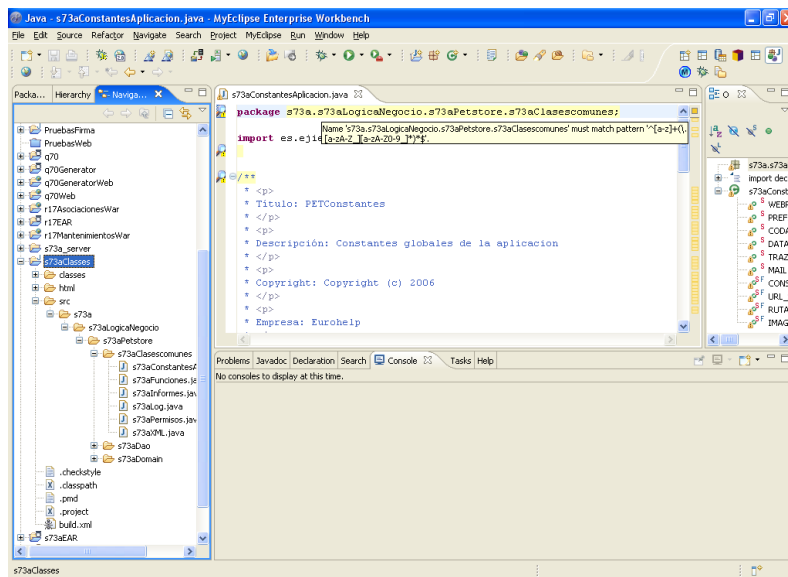


Al seleccionar dicha opción, emergerá una ventana con las características del proyecto al que pertenece. En esta ventana deberá aparecer la opción de configuración de Checkstyle.



Se selecciona entonces la configuración que se le quiere aplicar al proyecto y las opciones correspondientes a los ficheros pertenecientes al proyecto que serán chequeados (los ficheros a incluir y excluir). Una vez se han configurado las opciones oportunas, se activa la opción "checkstyle active for this Project" para que empiece a trabajar Checkstyle sobre dicho proyecto.

A partir de la activación del Checkstyle, tras cada compilación del proyecto, se podrá observar resaltado en el código los errores de estilo detectados por Checkstyle. El código erróneo se destacara en color amarillo y si se coloca el cursor del ratón encima de cualquier error se podrá ver el tipo y el mensaje asociado a dicho error.



4 Utilidad práctica

Checkstyle, por defecto y como ya se ha comentado, incorpora las recomendaciones de Sun sobre el estilo de código, pero estas reglas pueden ser redefinidas e incluso creadas completamente desde cero. Para el entorno de EJIE existe un fichero de reglas personalizado, que es el que se tiene que utilizar. Se puede descargar del SPS de CAC:

[http://elkarlan.ejje/webguneak/otc/Entregables/Repositorio%20Base/HERRAMIENTAS/Herramientas%20de%20Desarrollo%20\(PC\)/Plugins%20Análisis%20código%20estático/Calidad%202.0/reglas%20Checkstyle/OTC_Sonar_Checkconfig_5.3.xml](http://elkarlan.ejje/webguneak/otc/Entregables/Repositorio%20Base/HERRAMIENTAS/Herramientas%20de%20Desarrollo%20(PC)/Plugins%20Análisis%20código%20estático/Calidad%202.0/reglas%20Checkstyle/OTC_Sonar_Checkconfig_5.3.xml)

Se trata de un fichero checkconfig.xml. Guardar el fichero en un directorio en el puesto.

4.1 Aportaciones de Checkstyle

Checkstyle proporciona una completa colección de recomendaciones que se pueden aplicar a nuestros proyectos. Es capaz de validar distintos estándares de construcción (Javadocs, sentencias *import*, convenciones de nombres, etc.) y de encontrar problemas de diseño de clases, códigos duplicados o algunos bugs.

La referencia completa, organizada alfabéticamente, de estas recomendaciones se puede encontrar en el siguiente enlace <http://checkstyle.sourceforge.net/availablechecks.html>.

Siguiendo el criterio de funcionalidad se pueden considerar las siguientes agrupaciones:

- **Standard Checks:** Son aplicables al estilo general de programación Java y no requieren de librerías externas.
 - [Javadoc Comments](#): Entre otras funciones, comprueban la existencia de comentarios *Javadoc* (y si éstos están bien formados) en la definición de clases e interfaces, métodos o constructores, variables, etc.
 - [Naming Conventions](#): Comprueban si los distintos identificadores cumplen con expresiones regulares concretas.
 - [Headers](#): Comprueban, en base a expresiones regulares o no, que los archivos fuente comiencen con cabeceras concretas.
 - [Imports](#): Se realizan comprobaciones para que las sentencias *import* no usen la notación “*”, que no sean redundantes, etc.
 - [Size Violations](#): Restringen el número de instrucciones, comprueban instrucciones demasiado largas o archivos fuente demasiado extensos, etc.
 - [Whitespace](#): Entre otras funciones, se comprueba si la expresión de inicialización o la expresión de incremento de las instrucciones *for* está incluida o no. O bien, se comprueba qué se ha introducido entre el identificador de un método, constructor, llamada a método o invocación de un constructor y el paréntesis izquierdo de la lista de parámetros, es decir si están en la misma línea o se requiere un espacio, etc.
 - [Modifiers](#): Se comprueba el orden de los modificadores conforme a las recomendaciones de la especificación del lenguaje Java ([Java Language specification, sections 8.1.1, 8.3.1 and 8.4.3](#)), la existencia de modificadores redundantes en interfaces, etc.
 - [Block Checks](#): Comprueba la existencia de bloques vacíos y bloques anidados, la colocación de las llaves de apertura “{” y de cierre “}”, etc.
 - [Coding](#): Se realizan múltiples comprobaciones. Por citar algunas, se detectan sentencias vacías, existencia de ciertos literales numéricos no definidos como constantes, sentencias *switch* que no incluyen la cláusula *default*, etc.

- [Class Design](#): .Comprobaciones del diseño de clases. Por ejemplo, se comprueba la correcta visibilidad de los elementos de las clases, comprueba si las clases que sólo contienen constructores privados son declaradas como *final*, etc.
 - [Duplicate Code](#): Realiza una comprobación estricta de código duplicado incluyendo, para la comparación del código, comentarios *Javadoc*, líneas vacías entre métodos.
 - [Metrics](#): Comprobaciones que restringen el número de veces que se usan operadores "&&", "||" o "^" dentro de una misma instrucción, que limitan la complejidad ciclomática, etc.
 - [Miscellaneous](#): Permite realizar numerosas comprobaciones como son la existencia de instrucciones *System.out.println()* o *System.exit()*, la búsqueda de comentarios *TODO*, el estilo en la definición de los tipos *array*, etc.
- **Optional Checks**: Configuradas como submódulos de *TreeWalker* (ver apartado de *Creación de reglas de validación personalizadas*):
 - [J2EE Checks](#): Permite comprobar si un *entity bean* satisface ciertos requerimientos de clase, si todos los campos *static* están definidos como *final*, etc.

4.2 Aportaciones personalizadas

Puede resultar interesante crear reglas de validación personalizadas en los casos particulares que no se vean contemplados por las recomendaciones de Sun (por ejemplo, para particularizar para un estilo propio de programación):

- Comprobar que las llaves de apertura y de cierre siempre comiencen en una nueva línea.
- Comprobar que no se codifiquen dos instrucciones en una misma línea.
- Etc.

4.3 Creación de reglas de validación personalizadas

Cabe la posibilidad de que las reglas de validación proporcionadas por CheckStyle no cubran totalmente nuestras necesidades: En ese caso podremos crear nuestras propias reglas de validación.

No es necesario crear reglas de validación por los grupos de desarrollo. Ya existe en EJIE un fichero de las reglas personalizado, que es el que hay que utilizar con Checkstyle. Se puede descargar del SPS de CAC:

[http://elkarlan.ejie/webguneak/otc/Entregables/Repositorio%20Base/HERRAMIENTAS/Herramientas%20de%20Desarrollo%20\(PC\)/Plugins%20Análisis%20código%20estático/Calidad%202.0/reglas%20Checkstyle/OTC_Sonar_Checkconfig_5.3.xml](http://elkarlan.ejie/webguneak/otc/Entregables/Repositorio%20Base/HERRAMIENTAS/Herramientas%20de%20Desarrollo%20(PC)/Plugins%20Análisis%20código%20estático/Calidad%202.0/reglas%20Checkstyle/OTC_Sonar_Checkconfig_5.3.xml)

Se trata de un fichero *checkconfig.xml*. Guardar el fichero en un directorio en el puesto.

En primer lugar será necesario explicar cómo trabaja Checkstyle. CheckStyle transforma el código fuente en una representación en árbol, un elemento AST (Abstract Syntax Tree), que refleja la estructura del archivo. Esta transformación se realiza a través de un parseador denominado **ANTLR**.

La funcionalidad de Checkstyle está implementada en módulos que pueden ser conectados a Checkstyle. Estos módulos pueden contener, a su vez, a otros módulos formando una estructura en árbol. En esta estructura, los módulos de mayor nivel implementan la interfaz *FileSetCheck*. Checkstyle proporciona, por defecto, unas pocas implementaciones de la interfaz *FileSetCheck* y una de ellas es la clase *TreeWalker*. Esta

clase soporta submódulos que heredan de la clase *Check*. El objeto *TreeWalker* realizará la transformación al elemento AST (estructura en árbol) y llamará a los objetos *Check* encargados de realizar las diferentes validaciones.

Tras esta breve explicación del funcionamiento de Checkstyle podemos comenzar con el ejemplo de creación de una regla de validación personalizada que detectará el número máximo de métodos que puede tener una clase.

Los pasos a seguir son los siguientes:

1. En primer lugar habrá que implementar la clase que comprobará el número total de métodos existentes en una clase de (consideramos en el ejemplo que un único método sea el máximo admisible):

```
package com.pruebas.checks;

import com.puppycrawl.tools.checkstyle.api.*;

public class MethodLimitCheck extends Check
{
    private int max = 1;

    public int[] getDefaultTokens()
    {
        return new int[]{TokenTypes.CLASS_DEF, TokenTypes.INTERFACE_DEF};
    }

    public void visitToken(DetailAST ast)
    {
        // encontrar el nodo OBJBLOCK por debajo de CLASS_DEF/INTERFACE_DEF
        DetailAST objBlock = ast.findFirstToken(TokenTypes.OJBLOCK);

        // contar el número de hijos del elemento OBJBLOCK que sean del tipo METHOD_DEFS
        int methodDefs = objBlock.getChildCount(TokenTypes.METHOD_DEF);

        // informar del error en el caso de que el límite sea alcanzado
        if (methodDefs > max) {
            log(ast.getLineNo(), "too many methods, only " + max + " are allowed");
        }
    }
}
```

Al encontrar un elemento del tipo *CLASS_DEF* o del tipo *INTERFACE_DEF* el objeto *TreeWalker* llamará al método *visitToken* de esta clase que realizará las siguientes funciones:

- Se buscarán elementos del tipo *OBJBLOCK* que sean hijos de elementos del tipo *CLASS_DEF* o del tipo *INTERFACE_DEF*.
- Se contarán el número de hijos del elemento de tipo *OBJBLOCK* que sean elementos del tipo *METHOD_DEFS*.
- Se informará del error en el caso de que el límite máximo de métodos admisible por clase sea

alcanzado (en este caso concreto se considerará solamente uno).

En el caso de que se deseara que el número máximo admisible de métodos fuese un número configurable sólo habría que añadir un método *setter* (mutador) como el siguiente:

```
public void setMax(int limit)
{
    max = limit;
}
```

Esto funcionará para todos los tipos primitivos, para tipos String y arrays de todos estos.

2. En segundo lugar se creará el archivo XML de configuración de los estándares de validación en el que se incluirá la validación anteriormente implementada (por ejemplo, *mycustomrules.xml*):

```
<?xml version="1.0"?>
<!DOCTYPE module PUBLIC "-//Puppy Crawl//DTD Check Configuration 1.2//EN"
"http://www.puppycrawl.com/dtds/configuration_1_2.dtd">

<module name="Checker">
  <module name="TreeWalker">
    <module name="com.pruebas.checks.MethodLimitCheck"/>
  </module>
</module>
```

Finalmente se creará un **.jar** (por ejemplo *misReglas.jar*) que contenga la clase implementada junto a los ficheros de propiedades, en caso de que los hubiera.

4.3.1. Configuración de las reglas para el plug-in Checkstyle de Eclipse

Para configurar las reglas para **Eclipse** seguiremos los siguiente pasos (teniendo ya creados tanto la clase que implementará la validación como el fichero de configuración de estándares de validación):

1. Generaremos un fichero de descripción de paquetes *checkstyle_packages.xml* como el del ejemplo siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE checkstyle-packages PUBLIC "-//Puppy Crawl//DTD Package Names 1.0//EN"
"http://www.puppycrawl.com/dtds/packages_1_0.dtd">

<checkstyle-packages>
  <package name="com.pruebas.checks"/>
</checkstyle-packages>
```

2. También crearemos un fichero de metadatos *checkstyle-metadata.xml*, como el siguiente, para describir cada una de las clases de tipo Check creada:

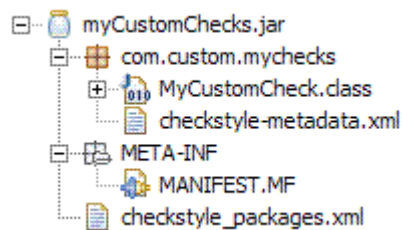
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE checkstyle-metadata PUBLIC "-//eclipse-cs//DTD Check Metadata 1.0//EN" "http://eclipse-
cs.sourceforge.net/dtds/checkstyle-metadata_1_0.dtd">

<checkstyle-metadata>
```



```
<rule-group-metadata name="Checks personalizados" priority="100">
  <rule-metadata name="MethodLimit" internal-name="MethodLimit" parent="TreeWalker">
    <alternative-name internal-name="com.pruebas.checks.MethodLimitCheck"/>
    <description>Verifica el numero de metodos de cada clase.</description>
  </rule-metadata>
</rule-group-metadata>
</checkstyle-metadata>
```

3. Crearemos un fichero **.jar** (por ejemplo *misReglas.jar*) que contenga la clase implementada junto a los ficheros de propiedades, en caso de que los hubiera, y agregaremos el fichero *checkstyle_packages.xml* en la raíz. También colocaremos el fichero *checkstyle-metadata.xml* con su respectivo *checkstyle-metadata_1_0.dtd* dentro del paquete que contiene la clase creada. La estructura del .jar se debe asemejar a la mostrada en la siguiente figura:



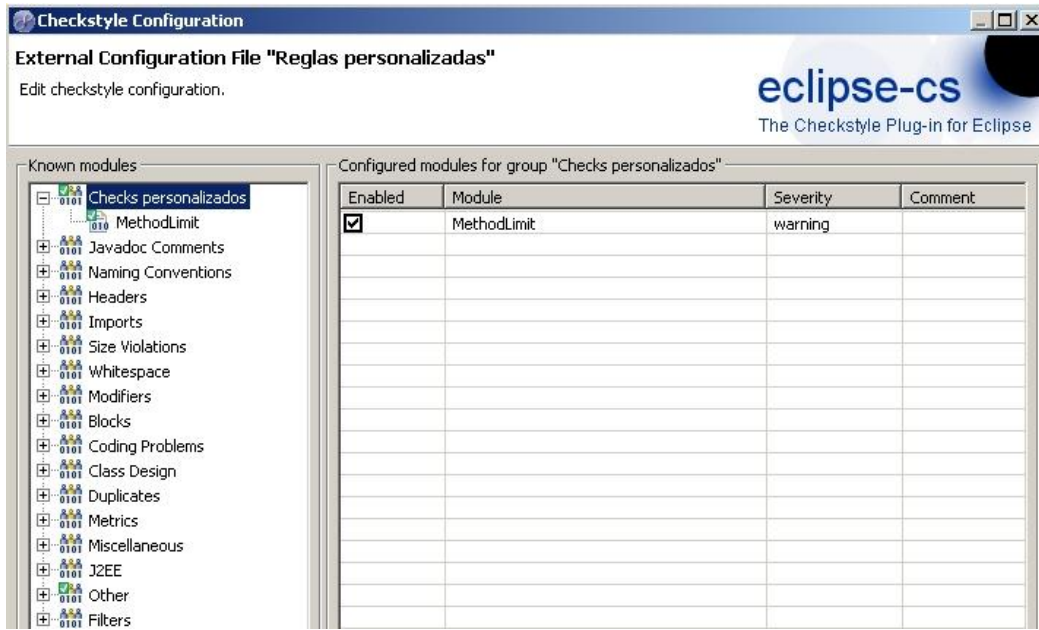
4. Nos dirigiremos al directorio:

```
<path-to-my-eclipse>/plug-ins/com.atlassw.tools.eclipse.checkstyle_x.x.x
```

5. Y copiaremos, en dicho directorio, el archivo *mycustomrules.xml* anteriormente creado.
6. A su vez, copiaremos el **jar** obtenido dentro del directorio *extension-libraries*.

De esta manera podremos configurar completamente a través del editor de configuración del plug-in las reglas de validación creadas. Puede encontrar más información en la siguiente dirección http://eclipse-cs.sourceforge.net/extending_custom_checks.html.

La siguiente imagen muestra la nueva configuración que hemos introducido:



5 Tareas ant en servidor

La validación del código java con Checkstyle también es posible ejecutarla en el servidor como tareas ant. Así se asumen los siguientes acuerdos:

- El fichero de configuración de Checkstyle se denomina “**checkconfig.xml**” y se ubica en “/dominio_wls8/j2se/jakarta-ant-1.5.3-1/rules”.
- Los ficheros de salida resultado de la validación serán, “**ckeckstyle_report.xml**” y “**ckeckstyle_report.html**”, en formato XML y HTML respectivamente, y se ubicarán en “/aplic/xxx/java/javs/xxxEAR/aaaTestingWar/test/informes”. (Siendo xxx el código de aplicación).
- La tarea que ejecuta Checkstyle se denomina “**checkstyle**”.

ant checkstyle

Este ejemplo realiza la verificación del código de *s73aPetStoreWar* a partir del archivo de reglas *checkconfig.xml* y muestra los resultados obtenidos a través de un informe *.rtf* y a través de un gráfico en formato *.png*.

El primer paso sólo es necesario para trabajar con la interfaz más adecuada para la verificación del estilo de código por parte de Checkstyle.

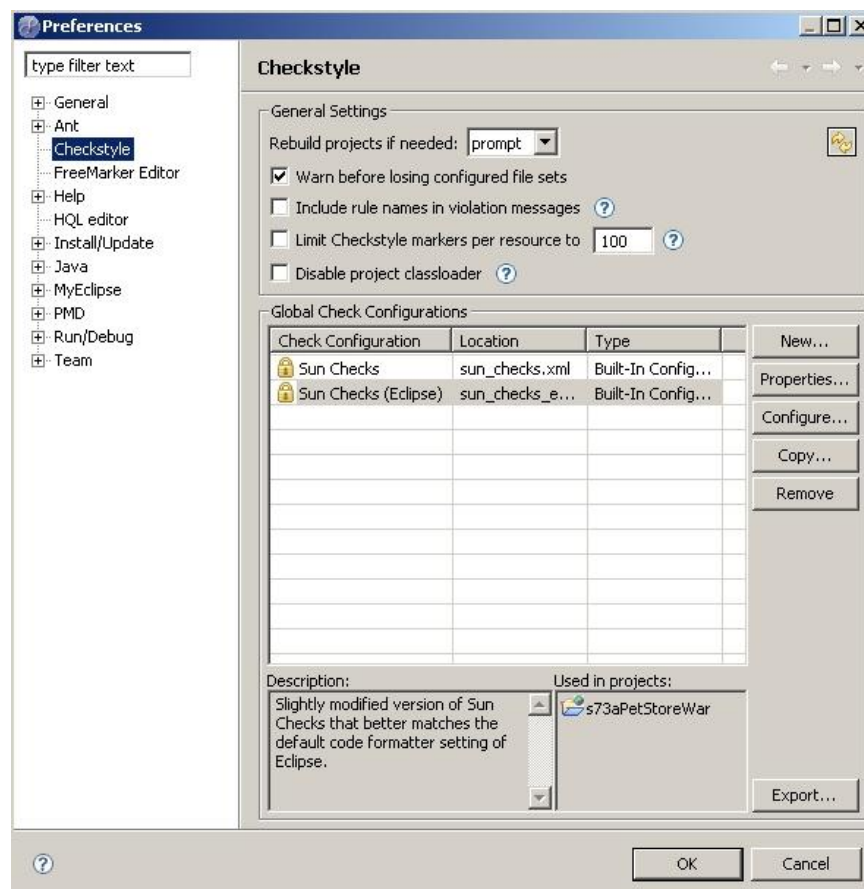
-
- The screenshot shows the 'Show View' dialog box. The 'type filter text' field is empty. The tree view displays a list of views, with 'Checkstyle' selected. Under 'Checkstyle', three sub-items are listed: 'Checkstyle violations', 'Checkstyle violations chart', and 'Duplicated Code'. The 'OK' button is highlighted with a red rectangle.

[illegible]

- Es el turno de configurar el conjunto de las reglas a utilizar con Checkstyle. Para ello se selecciona la opción del menú principal *Window > Preferences...*



Se abrirá la ventana de preferencias que aparece en la imagen siguiente.

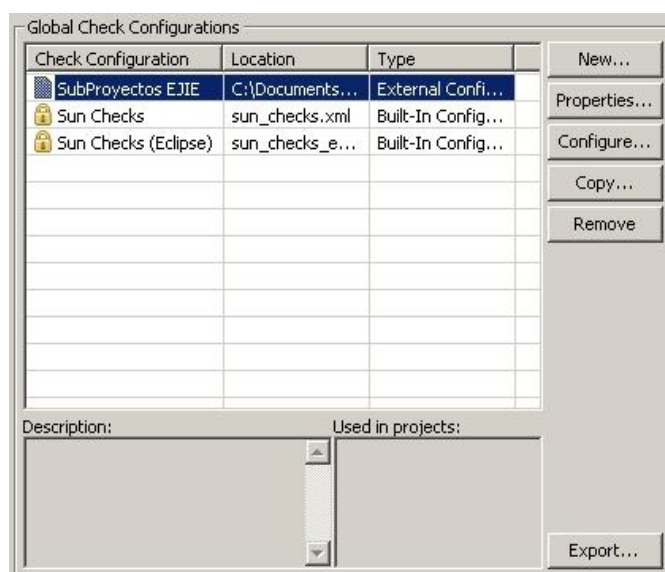


Dentro de esta ventana se puede observar que Checkstyle, por defecto, incorpora las recomendaciones de *Sun* sobre el estilo de código (*Sun Checks*). Además también incorpora una versión modificada de estas recomendaciones de *Sun* que encaja mejor con la configuración del “formateador” de código de Eclipse.

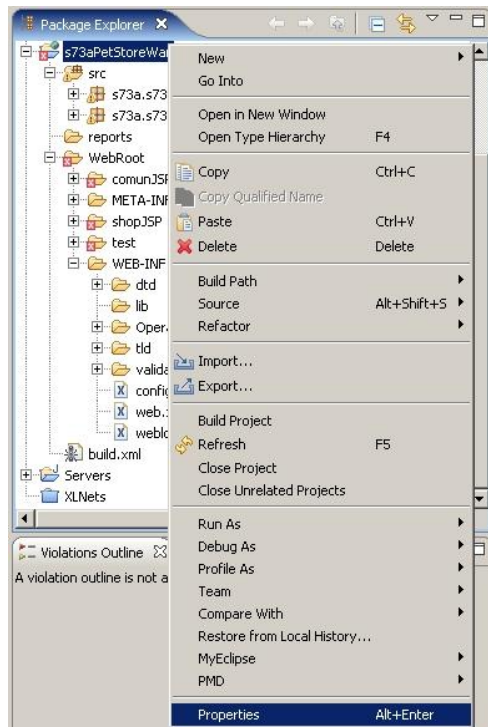
3. Se añadirá el nuevo conjunto de reglas que se desea utilizar para la verificación del código. Para ello se pulsará el botón New... Se abrirá la ventana “Check Configuration Properties” desde la cual se podrá acceder al archivo de reglas deseado.



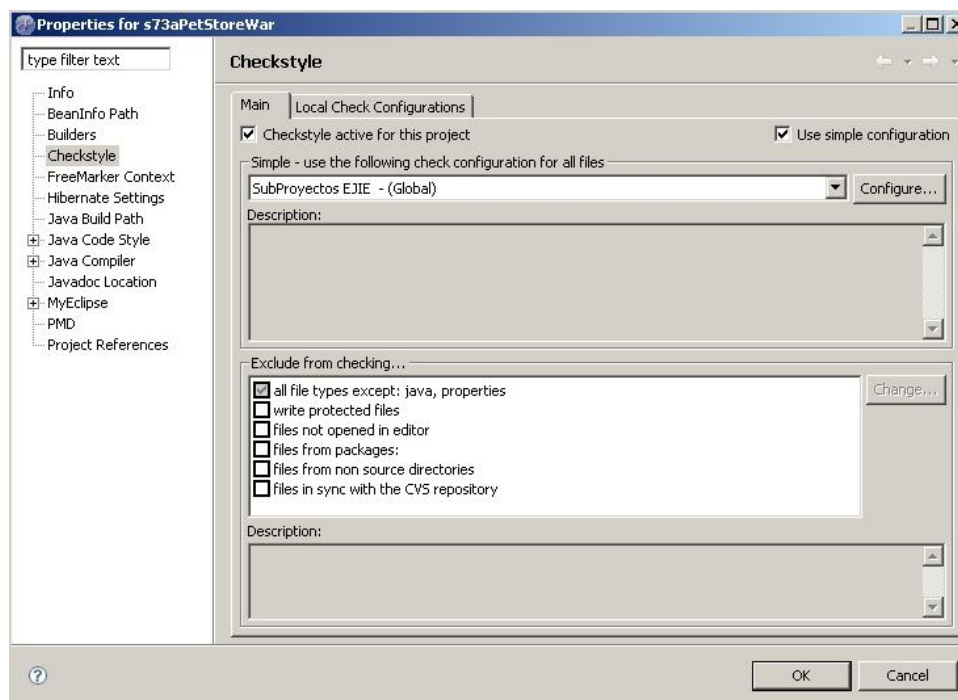
Para ello, se seleccionará el tipo External Configuration File y se localizará el archivo de reglas (*checkconfig.xml*) haciendo uso del botón “Browse...” o bien escribiendo directamente la ruta en el campo *Location*.



4. Ahora se podrá habilitar la ejecución de la verificación del código por parte de Checkstyle por separado para cada proyecto a la hora de realizar la construcción de los mismos.

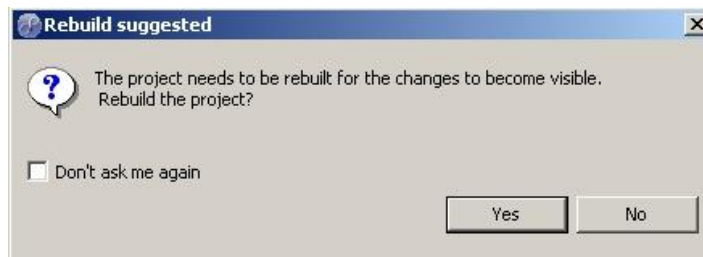


En este ejemplo se habilitará Checkstyle para el proyecto *s73aPetStoreWar*.

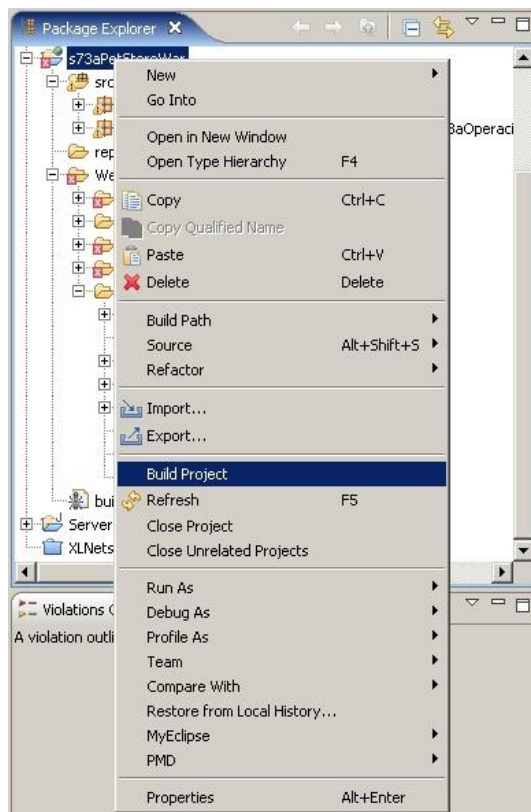


Entonces, se seleccionará dicho proyecto y a través de su menú contextual (botón derecho del ratón) se seleccionará la opción *Properties* como muestra la imagen superior. Se mostrará la ventana *Properties for s73aPetStoreWar*, ventana que muestra las propiedades del proyecto seleccionado, y se activará la casilla de verificación *Checkstyle active for this project*.

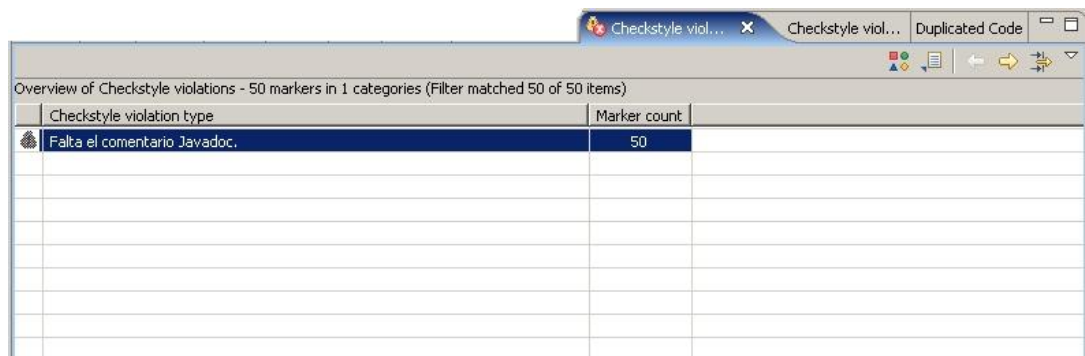
5. En este momento sólo quedará reconstruir el proyecto para que se lleve a cabo la verificación del código. Se pulsará el botón *OK*. Entonces, o bien Checkstyle mostrará el siguiente mensaje para encargarse de realizar la reconstrucción automática del proyecto:



O bien habrá que realizar a mano la reconstrucción. Para ello habrá que seleccionar la opción del menú contextual *Build Project* relativa al proyecto seleccionado:

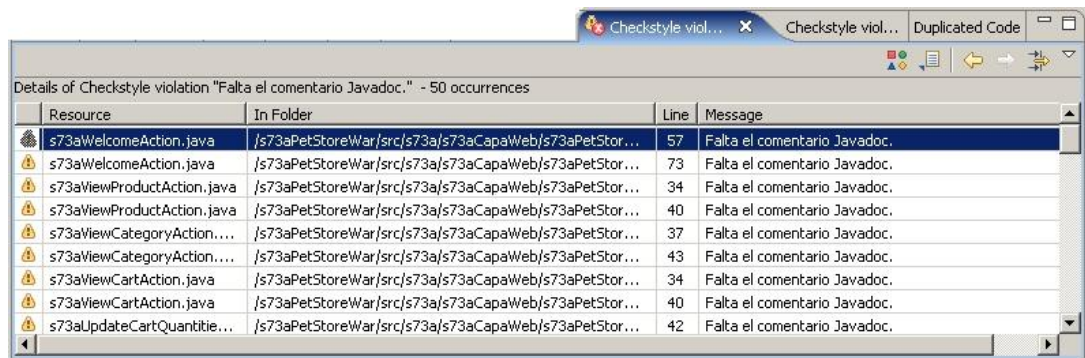


6. Tras reconstruir, se podrá obtener una vista general de los diferentes tipos de violaciones de las reglas obtenidos tras la verificación del código en la vista *Checkstyle Violations* (en caso de que las hubiera) como muestra la figura siguiente:



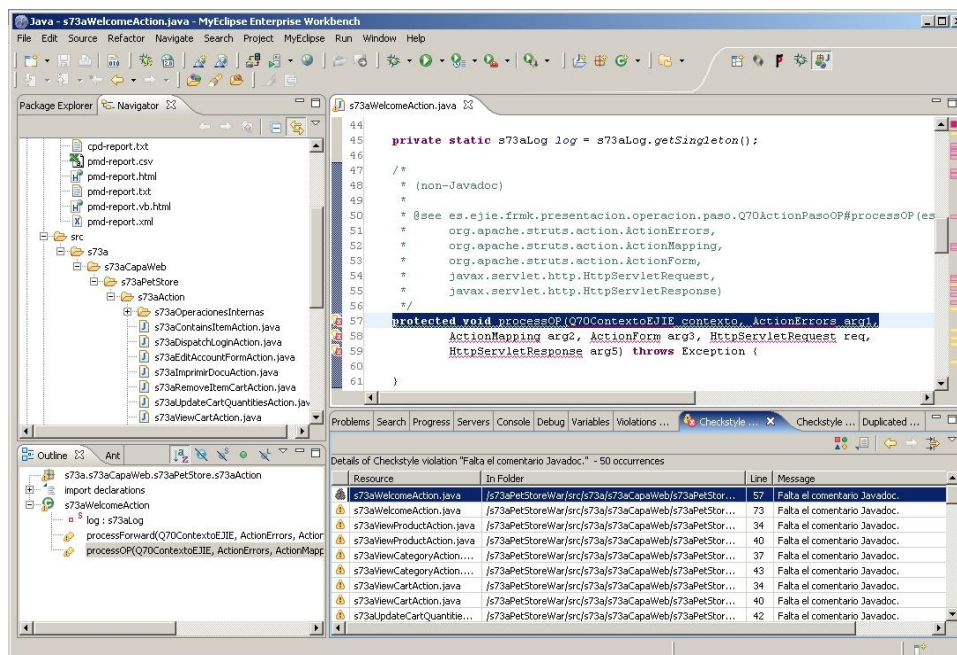
| Checkstyle violation type | Marker count |
|------------------------------|--------------|
| Falta el comentario Javadoc. | 50 |

Se puede observar que el único tipo obtenido es “Falta el comentario Javadoc”. Desplegando este tipo se muestra un listado con los detalles de las diferentes localizaciones en las que se ha detectado este tipo de violación.



| Resource | In Folder | Line | Message |
|----------------------------|--|------|------------------------------|
| s73aWelcomeAction.java | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStor... | 57 | Falta el comentario Javadoc. |
| s73aWelcomeAction.java | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStor... | 73 | Falta el comentario Javadoc. |
| s73aViewProductAction.java | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStor... | 34 | Falta el comentario Javadoc. |
| s73aViewCategoryAction.... | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStor... | 40 | Falta el comentario Javadoc. |
| s73aViewCartAction.java | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStor... | 37 | Falta el comentario Javadoc. |
| s73aViewCartAction.java | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStor... | 43 | Falta el comentario Javadoc. |
| s73aUpdateCartQuantitie... | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStor... | 34 | Falta el comentario Javadoc. |
| s73aUpdateCartQuantitie... | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStor... | 40 | Falta el comentario Javadoc. |
| s73aUpdateCartQuantitie... | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStor... | 42 | Falta el comentario Javadoc. |

Pulsando sobre cada una de ellas se podrá acceder a la instrucción que ha generado la violación.



The screenshot shows the Eclipse IDE with the file `s73aWelcomeAction.java` open. The code snippet below is highlighted, showing a Javadoc violation. The violation message is "Falta el comentario Javadoc." (Missing Javadoc comment).

```

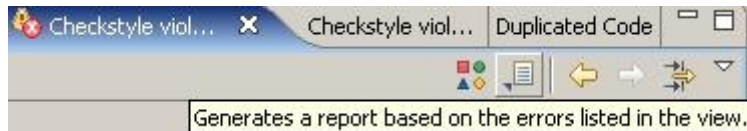
private static s73aLog log = s73aLog.getSingleton();

/*
 * (non-Javadoc)
 * @see es.ejje.frmk.presentation.operacion.paso.Q70ActionPasoOP#processOP(es
 * org.apache.struts.action.ActionErrors,
 * org.apache.struts.action.ActionMapping,
 * org.apache.struts.action.ActionForm,
 * javax.servlet.http.HttpServletRequest,
 * javax.servlet.http.HttpServletResponse)
 */
protected void processOP(Q70ContextoEJJE contexto, ActionErrors arg1,
    ActionMapping arg2, ActionForm arg3, HttpServletRequest req,
    HttpServletResponse arg5) throws Exception {
    ...
}

```

The bottom of the screenshot shows the "Details of Checkstyle violation" window, which lists the violations found in the project, including the one in `s73aWelcomeAction.java`.

7. Para generar un informe *.rtf* habrá que pulsar el icono correspondiente de la vista *Checkstyle violations* como muestra la siguiente imagen.



Abriendo el informe con la herramienta Microsoft Word, éste mostrará el siguiente aspecto:

Checkstyle statistics

Overview of Checkstyle violations - 50 markers in 1 categories (Filter matched 50 of 50 items)

| Checkstyle violation type | Marker count |
|------------------------------|--------------|
| Falta el comentario Javadoc. | 50 |

Details of Checkstyle violation "Falta el comentario Javadoc." - 50 occurrences

| Resource | In Folder | Line | Message |
|-----------------------------|---|------|------------------------------|
| s73aWelcomeAction.java | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStore/s73aAction | 57 | Falta el comentario Javadoc. |
| s73aWelcomeAction.java | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStore/s73aAction | 73 | Falta el comentario Javadoc. |
| s73aViewProductAction.java | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStore/s73aAction | 34 | Falta el comentario Javadoc. |
| s73aViewProductAction.java | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStore/s73aAction | 40 | Falta el comentario Javadoc. |
| s73aViewCategoryAction.java | /s73aPetStoreWar/src/s73a/s73aCapaWeb/s73aPetStore/s73aAction | 37 | Falta el comentario Javadoc. |

8. Accediendo a la vista *Checkstyle violations chart* y pulsando el icono Display categories < 1% se podrá acceder a un gráfico que mostrará porcentualmente los tipos de violaciones que se han producido.



Existe la posibilidad de guardar dicho gráfico en formato .png. Sólo habrá que hacer uso del icono correspondiente:

