Práctica 1

Pruebas estáticas de código con PMD

Objetivo
Conocer el *plug-in* para revisiones estáticas de código:

- **PMD**:
- https://sourceforge.net/projects/pmd/files/pmd-eclipse/zipped/net.sourceforge.pmd.eclipse.p2updatesite-4.3.0.v20190428-0918.zip/download
- https://docplayer.es/6265239-Pmd-manual-de-usuario.html

Previo

1) Descarga el fichero de la práctica 1 del aula virtual. Contiene el proyecto P1-pmd.
2) Descomprime el proyecto en el espacio de trabajo e importa los proyectos en Eclipse.
3) Una vez importado el proyecto a eclipse se debe renombra para indicar tu nombre y apellidos de la siguiente forma: *P1-pmd-Apellidos_Nombre*.

PMD (*Programming Mistake Detector*)

PMD es una herramienta de análisis de código que busca problemas potenciales como:

- Posibles errores: sentencias `try/catch/finally/switch` vacías.
- Código muerto: variables locales, parámetros o método privados no utilizados.
- Código mejorable: uso de `String`/`StringBuffer` ineficiente.
- Expresiones excesivamente complicadas: sentencias `if` innecesarias, bucles `for` que podrían ser bucles `while`.
- Código duplicado: código copiado/pegado significa errores copiados/pegados.

El *plugin* se puede **configurar** a través de la opción de menú `Ventana >> Preferencias…` En el árbol de la izquierda se selecciona PMD y aparecerá la ventana de configuración de la Fig.1. Podemos ver que los errores están clasificados en cinco niveles de prioridad, cada uno de ellos identificado por un símbolo, un color y un nombre, que se pueden modificar.

Desplegando el árbol de la izquierda en la opción PMD, podemos acceder a las opciones de configuración de las reglas (`Rule Configuration`, Fig. 2) que permite gestionar las reglas que se van a aplicar en nuestros proyectos. Se pueden activar y desactivar las reglas, y seleccionando cada una de ellas, se puede modificar sus propiedades desde la pestaña `Rule` en la Fig. 2, por ejemplo, la prioridad. Una descripción completa de las reglas soportadas se puede encontrar en: http://pmd.sourceforge.net/pmd-4.3.0/rules/index.html

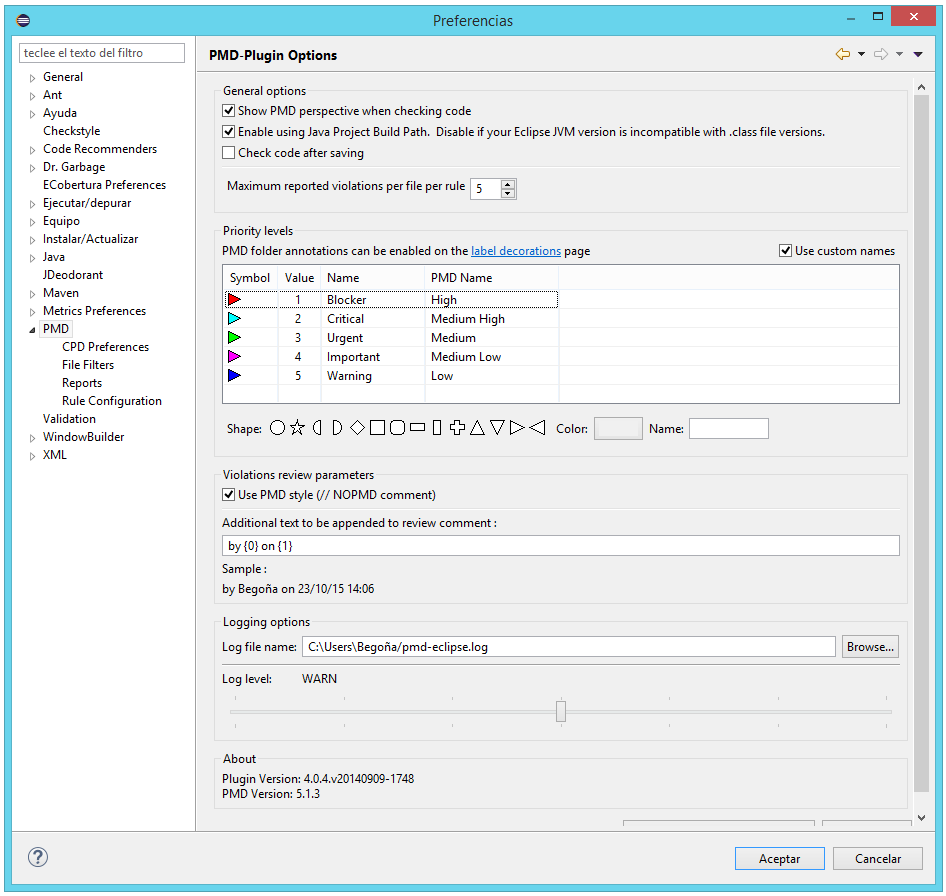

Fig. 1 Configuración de PMD.

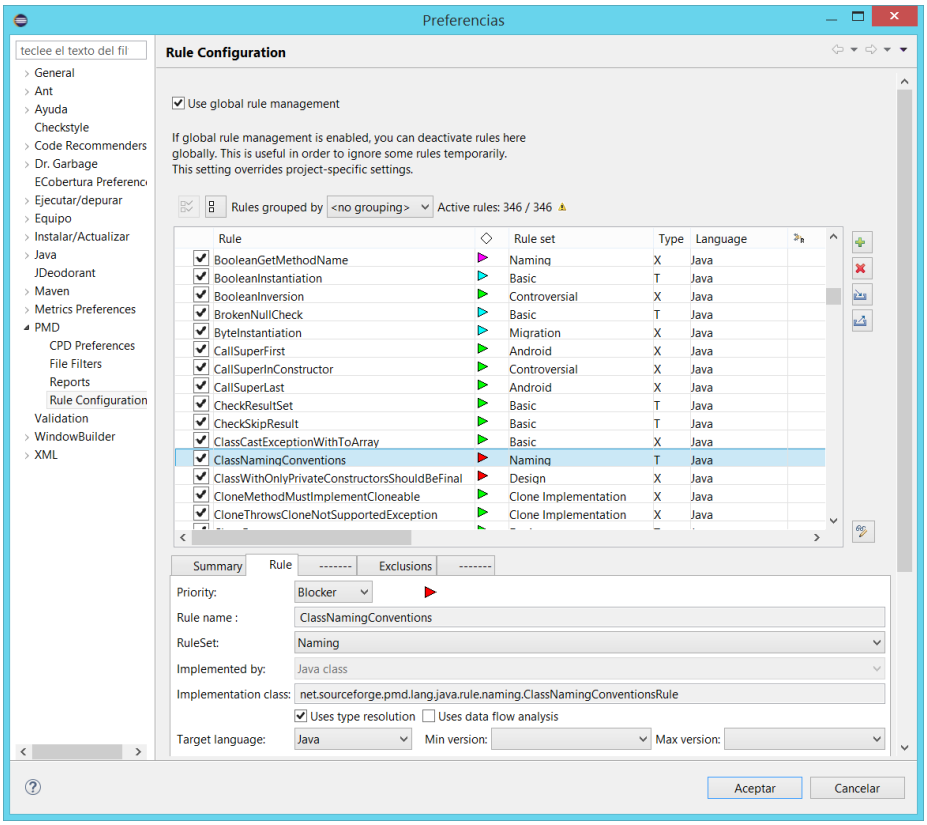

Fig. 2 Configuración de reglas.

El chequeo de código con PMD se puede activar desde el menú contextual del proyecto (botón derecho del ratón sobre el proyecto) y seleccionando PMD en el árbol de la izquierda (Fig.3). Activamos la opción `Enable PMD`. Para el chequeo se puede seleccionar un conjunto de trabajo (*working set*) que no es más que seleccionar qué ficheros y subdirectorios van a ser chequeados. Si no se selecciona ningún conjunto de trabajo se chequea el proyecto actual. También se pueden activar y desactivar reglas para este proyecto.

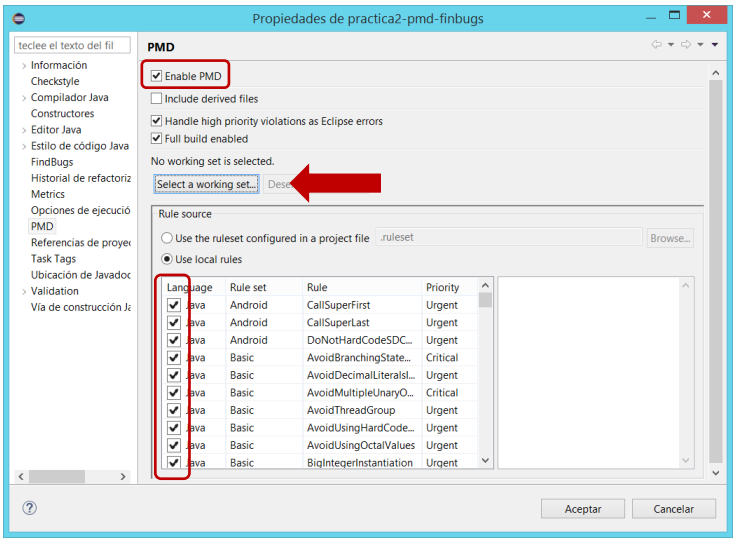

Fig. 3 Ventana de propiedades del proyecto.

Para **ejecutar** el chequeo del código con el *plugin* PMD, seleccionamos el proyecto y pulsamos el botón derecho del ratón. Seleccionando la opción de menú PMD, se despliegan las opciones que se muestran en la Fig. 4, entre las que marcamos la opción `Check Code`.

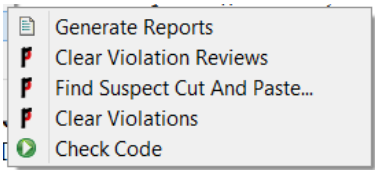

Fig. 4 Menú opciones PMD.

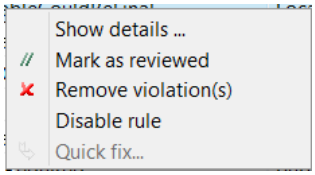

Fig. 5 Menú contextual de la vista Violations Outline.

El resultado de la ejecución se mostrará en una nueva perspectiva del entorno (Fig. 6), si es que se ha activado está opción en la configuración del *plugin* (ver Fig.1 Opciones generales). Esta perspectiva tiene dos vistas asociadas (que se pueden abrir en cualquier momento desde el menú `Ventana >> Mostrar vista >> Otras …`):

- *Violations Outline*: Muestra los errores encontrados en la clase seleccionada en el Explorador de paquetes. Haciendo doble *click* sobre cada error te lleva a la línea de código donde se encuentra. Por otro lado, pulsando con el botón derecho del ratón sobre cada uno de los errores aparece el menú contextual de la Fig. 5, desde donde se puede, entre otras cosas:
    - Obtener los detalles de la regla violada (`Show details…`).
    - Introducir un comentario en el código para que ignore esa violación (`Mark as reviewed`). Este comentario indica que el error ha sido revisado y puede no tenerse en cuenta. En los informes, por defecto, se ignoran las violaciones anotadas como revisadas. El formato de la etiqueta se puede configurar (ver Fig. 1), por defecto se mostrará `// NOPMD by {autor} on {date}`.

Es importante anotar quién ha decidido ignorar ese error e incluso sería interesante anotar el motivo.

- *Violations Overview*: Muestra un resumen de los errores encontrados en cada elemento del proyecto, agrupados por tipo de error. Para cada error se especifica el número de veces que se ha encontrado. Por ejemplo, en la Fig. 6, en la clase `Circulo` se han encontrado 8 violaciones de la regla que detecta que falta un comentario Javadoc (`CommentRequired`). La información que se muestra en esta vista se puede filtrar por el tipo de error seleccionando los triángulos de colores que identifica cada nivel de prioridad, para que sólo se muestren, por ejemplo, los de prioridad alta o media alta.

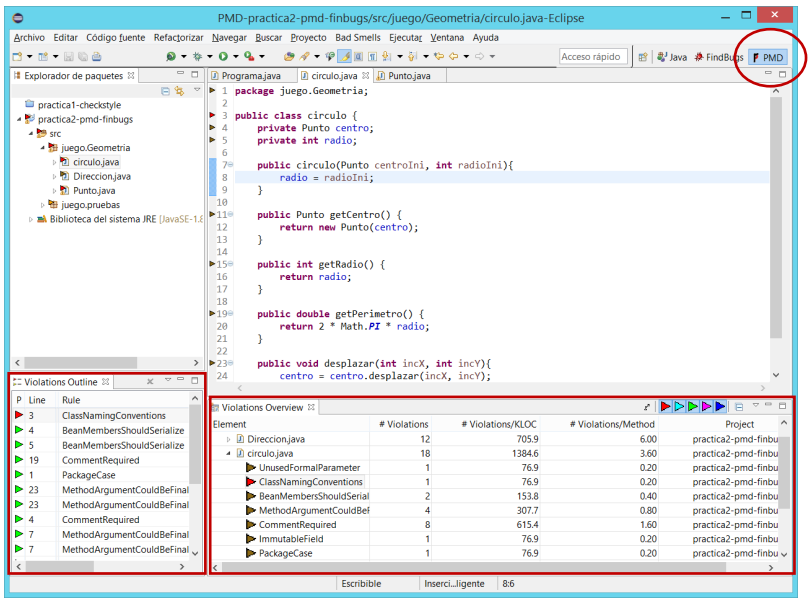

Fig. 6 Perspectiva PMD y vistas asociadas.

Además del chequeo del código, el *plugin* nos ofrece otras funcionalidades a través del menú contextual (Fig. 4). Estas opciones son:

- **Generar informes**: guarda en un documento el resultado del chequeo del código. El formato de los informes generados se puede configurar (ver Fig. 1 y 2 la opción `Reports` en el árbol de la izquierda bajo `PMD`). Por defecto, los informes se generan en documentos de texto (fichero `pmd-report.txt`) que se almacena en una carpeta denominada `reports` dentro del proyecto. En el informe, los resultados se muestran en cuatro columnas: número incremental que identifica el problema, clase para la que se ha detectado el problema, número de línea dentro del fichero y la descripción del problema.
- Borrar los comentarios de revisión de las reglas, esto es, los comentarios `// NOPMD` (`Clear Violation Reviews`).
- Encontrar código sospechoso de haber sido copiado y pegado. El resultado del análisis se muestra en una vista y en un fichero (si se ha marcado esta opción) que se guardará en la carpeta `reports`. El formato del fichero puede ser textual o xml. El informe muestra el código que aparece duplicado (dentro de la misma clase o en varias clases distintas) e indica la clase a la que pertenece este código y la línea a partir de la cual se encuentra implementado.
- Eliminar las marcas de chequeo del código (`Clear Violations`).

Ejercicios

1) Abre la ventana de preferencias (menú `Ventana >> Preferencias …`), selecciona PMD en el panel de la izquierda y marca la casilla "Check code after saving", para que se actualice el cheque tras los cambios.

2) En la ventana de preferencias, selecciona del menú PMD de la izquierda "Rule Configuration", marca "Use global rule management" y agrupa las reglas por "Conjunto de reglas" (`Rules grouped by >> Rule set`). Ver Fig. 2.

3) **Activa** los siguientes conjuntos de reglas: `Basic`, `Braces`, `Clone Implementation`, `Code Size`, Complexity, `Controversial`, `Coupling`, `Design`, `Empty Code`, `Import Statements`, `Java Logging`, `Naming`, `Optimiation`, `String and StringBuffer`, `Style`, `Unnecessary`, `Unused Code`. Estos grupos son los configurados en la versión instalada en los laboratorios de practicas de la asignatura.

   IMPORTANTE: Si las prácticas se realizan con otra versión del plugin, se deben marcar las reglas contenidas en el archivo complementario de la tarea *Reglas_No_Lab.txt*

   NOTA: el conjunto de reglas `Comments` (abre el grupo para ver las reglas que incluye) se refiere a los comentarios de documentación que en un proyecto real no deberían desactivarse.

4) **Desactiva** las siguientes reglas:
   - Del conjunto `Clone Implementation` → `CloneThrowsCloneNotSupportedException`.Del conjunto `Controversial` → `OnlyOneReturn y DataflowAnomalyAnalisis.`
   - Del conjunto `Optimization` → `LocalVariableCouldBeFinal` y `MethodArgumentCouldBeFinal`.

5) Chequea el código de tu proyecto con PMD y revisa las violaciones detectadas.

6) Anota como revisado (`// Mark as reviewed`):
   - En la clase `Punto`, las violaciones que tienen que ver con el error de que las variables tienen nombres muy cortos, justificando el motivo de la omisión. También la violación de la Ley de Demeter en el método toString.
   - En la clase `Programa`, la posible violación de la Ley de Demeter y la llamada a mostrar un mensaje en la consola.

7) **Genera un informe** de errores (el archivo se crea en la carpeta *reports* del espacio del proyecto) y **cambia su nombre** a *pmd-report-previo.txt*. **Comprueba** que las violaciones comentadas no aparecen en el informe.

   NOTA: Genera los informes después de haber chequeado el código de nuevo.

8) Limpia el código para eliminar el resto de violaciones detectadas por PMD.

9) Crea una copia de la clase `Punto` y ejecuta la opción del menú PMD para encontrar código sospechoso de haber sido copiado y pegado (debes seleccionar la codificación java). De forma automática debe generarse el informe.

Entregables

La entrega se realizará en la tarea del Aula Virtual asignada a la práctica con fecha de entrega 14/11/2022 23:55 horas y se debe subir a la tarea del Aula Virtual el espacio de trabajo de

Eclipse con el que se ha trabajado comprimido donde debe aparecer, además de los ficheros del proyecto (incluidas las configuraciones), los reportes generados por el plugin PMD solicitados. Para ello basta con comprimir la carpeta del espacio de trabajo Eclipse con toda la información generada durante la práctica y subir el fichero comprimido a la tarea del aula virtual.

IMPORTANTE: Al realizar la entrega se debe indicar en el editor en línea si la práctica se ha realizado en los ordenadores de los laboratorios o en un ordenador personal con una versión distinta del *plugin PMD*

La entrega estará abierta desde la fecha de apertura de la tarea (26/09/2022) hasta la fecha de entrega el 14/11/2022.