
J-Static-Analizer

J-Static-Analizer

Documento de Arquitectura de Software

Versión 1.0

Curso: Tecnología de Objetos	Versión: 1.0
J-Static-Analizer	Fecha: 28/11/2008
Documento de Arquitectura de Software	
http://j-static-analizer.blogspot.com/	

Historial de Revisiones

Fecha	Versión	Descripción	Autor	Revisado	Nota
20/11/2008	1.0	Versión inicial	<ul style="list-style-type: none"> Christian Portilla Pauca Yvan Florez Mayorga 		

Historial de Observaciones

Fecha	Versión	Observaciones

Curso: Tecnología de Objetos	Versión: 1.0
J-Static-Analizer	Fecha: 28/11/2008
Documento de Arquitectura de Software	
http://j-static-analizer.blogspot.com/	

Tabla de Contenido

1. Introducción	4	
1. Propósito	4	
2. Alcance	4	
3. Definiciones, Acrónimos y Abreviaciones	4	
4. Referencias	5	
5. Objetivos y restricciones de la arquitectura	6	
Objetivo	6	
Restricciones	6	
6. Vista de Casos de Uso	7	
7. Vista Lógica	8	
1. Descripción de los Subsistemas	8	
a. AnálisisEngine		8
b. StaticRules		8
2. Subsistemas	8	
8. Vista de Despliegue	9	
1. Introducción	9	
2. Diagramas De Despliegue	9	
3. PCStaticEvaluator	9	
4. Arquitectura del Proyecto	10	
JStaticAnalizer	11	
JStaticEngine	11	
JRules	11	

Curso: Tecnología de Objetos	Versión: 1.0
J-Static-Analizer	Fecha: 28/11/2008
Documento de Arquitectura de Software	
http://j-static-analizer.blogspot.com/	

Documento de Arquitectura de Software

1. Introducción

1. Propósito

Nuestro proyecto fue desarrollado en función de patrones de diseño de software para permitir que nuestro framework inicial para el desarrollo del proyecto final cumpla con las condiciones de evaluación de código de java de manera estática.

El propósito básico de nuestro plugin es en base a código java mejorar el desempeño y calidad de la programación en este lenguaje mediante la evaluación del código java de forma estática para determinar errores de lógica de programación antes de hacer una compilación final en un proyecto en particular.

2. Alcance

En este documento nos centramos en aspectos como son el desarrollo y la estructura propia del framework a ser desarrollado para la elaboración del plugin de evaluación estática de código java.

En nuestro framework se desarrolla componentes para cada parte de la evaluación de código en base a las 15 reglas a ser evaluadas.

El framework se elaborara en función de una revisión del código en aspectos como: Optimización de código, seguridad, gestión de memoria, código inseguro, métricas, convenciones de código, POO, entre otras.

3. Definiciones, Acrónimos y Abreviaciones

- Patrón de diseño.-
Es la solución a un problema básico de programación que es útil para múltiples ámbitos de programación.
- Evaluación estática.-
Es la evaluación de código antes de ser compilada para poder determinar errores en el código tales como malas prácticas de programación.
- Framework.-
Es un marco de programación el cual puede tener un ámbito de aplicación específico para elaborar programas de una clase específica.
- Métricas.-
Son evaluaciones al software en cuanto a la cantidad de líneas de código y relativo a cantidad de trabajo útil en la elaboración de un software en específico.
- Eclipse.-
Ide de Programación para java.
- Tokens.-
Tokens cada uno de los símbolos en un código analizados para su posterior evaluación tanto sintáctica como semánticamente u de otro clase.
- Analyzer.-
Analizador, en nuestro aplicativo de plugin para eclipse sera usado en el análisis del código.

Curso: Tecnología de Objetos	Versión: 1.0
J-Static-Analizer	Fecha: 28/11/2008
Documento de Arquitectura de Software	
http://j-static-analizer.blogspot.com/	

4. Referencias

- O. Agesen. Constrained-based type inference and parametric polymorphism. In b. Le charlier, editor, proceedings of the first international symposium on static analysis (sas'94), number 864 in lncs, namur, september 1994. Springer- verlag.
- B. Blanchet. Escape analysis for object oriented languages. Application to java. In proceedings of the conference on object-oriented programming, systems, languages and applications static analysis (oopsla'99), november 1999.
- J_gosling_b_joy_and_g_steele_thejavatm language specification_ addison_wesley_
- B.j.gosling and g_steele_java specification language technical report available from <http://www.javasoft.com>
- A simple semantics and static analysis for java security anindya banerjee and david a. Naumann stevens institute of technology, cs report 2001-1
- Finding security vulnerabilities in java applications with static analysis v. Benjamin livshits and monica s. Lam
- Computer science department stanford university {livshits, lam}@cs.stanford.edu
- Banatre, j. P., bryce, c., le metayer, d. "compile-time detection of information flow in sequential programs." In proc. Third european symp. Research in computer security, pages 55-73, volume lncs 875, brighton, uk, nov 1994. Springer-verlag.
- Thinking in java, 2nd edition, release 11 to be published by prentice-hall mid-june, 2000 bruce eckel, president,
- Mindview, inc. Prentice hall upper saddle river, new jersey 07458 www.phptr.com

Curso: Tecnología de Objetos	Versión: 1.0
J-Static-Analizer	Fecha: 28/11/2008
Documento de Arquitectura de Software	
http://j-static-analizer.blogspot.com/	

5. Objetivos y restricciones de la arquitectura

➤ **Objetivo**

El objetivo base de elaboración del framework es la revisión del código fuente en aspectos como: Optimización de código, seguridad, gestión de memoria, código inseguro, métricas, convenciones de código, POO, entre otras, para en base a estos poder identificar posibles malas prácticas de programación orientada a objetos y posibles excepciones sobre código fuente Java e informar al programador sobre estos problemas antes de generar los bytecodes.

➤ **Restricciones**

El framework no debe ser dependiente del número de reglas implementadas a ser usadas por el motor de verificación de código.

Las reglas a ser implementadas tienen niveles de severidad divididas en cuatro niveles: crítico, alto, medio, y bajo.

El framework para JStaticAnalizer debe poder ser instalado en Eclipse en modo plugin, y debe ser accesible por medio de un menú en eclipse con opciones de iniciar, detener, limpiar JSA, y analizar reglas.

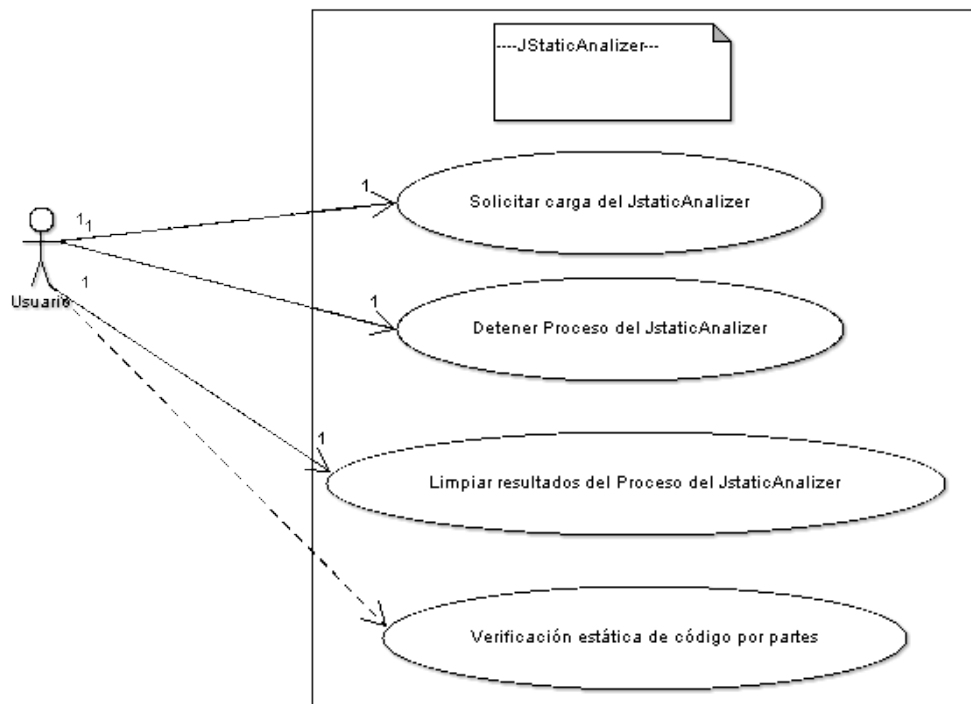
Los resultados deberían ser mostrados en el mismo IDE Eclipse, indicando el tipo de warning (POO o Excepción), el número de regla y el número de línea de código.

Curso: Tecnología de Objetos	Versión: 1.0
J-Static-Analizer	Fecha: 28/11/2008
Documento de Arquitectura de Software	
http://j-static-analizer.blogspot.com/	

6. Vista de Casos de Uso

Para el diseño del Framework JStaticAnalizer, se identifican como los casos de uso relevantes desde el punto de vista de la arquitectura, los abajo mencionados:

1. Solicitar carga del JstaticAnalizer
 - a. Esta representa la función principal del análisis de código
 - b. Para la misma necesitamos la precarga del plugin en eclipse para poder hacer un fácil uso de nuestro framework.
 - c. Este componente tiene que manejar la complejidad de evaluación de las reglas de análisis estático.
2. Detener Proceso del JstaticAnalizer
 - a. Esta función representa la función de detener el proceso de evaluación en un momento dado.
 - b. La complejidad relacionada con esto es poder hacer uso de los threads de proceso de carga del analizador estático.
3. Limpiar resultados del Proceso del JstaticAnalizer
 - a. Se encarga e limpiar resultados y cerrar ventanas fruto de la evaluación estática del código
4. Verificación estática de código por partes
 - a. Esta representa otro de los usos del framework como puede ser la carga del framework con alguna regla en particular para un trozo de código.
 - b. Este componente debe funcionar junto con el plugin en eclipse para facilitar el uso del mismo al usuario.



Curso: Tecnología de Objetos	Versión: 1.0
J-Static-Analizer	Fecha: 28/11/2008
Documento de Arquitectura de Software	
http://j-static-analizer.blogspot.com/	

7. Vista Lógica

Aquí podemos señalar que nuestro framework esta siendo elaborado en función a dos paquetes base como son:

1. AnalisisEngine
2. StaticRules

Estos como se puede detallar se encargan de las funciones principales encargadas al sistema como son: Evaluación estática, y elaboración del conjunto de reglas a evaluar por el sistema.

1. Descripción de los Subsistemas

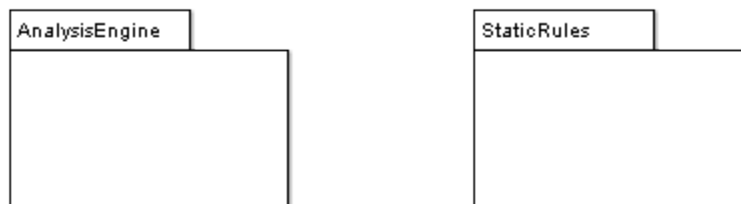
a. AnalisisEngine

Este es el motor de evaluación de código, este se encarga de cargar las definiciones de las reglas básicas necesarias para la evaluación. Esto lo hace primero haciendo un parsing de todos y cada uno de los términos o tokens de java, luego procesando su sintaxis y semántica, para luego poder evaluar las reglas necesarias que se tomen a evaluar en un determinado momento.

b. StaticRules

Este componente se encarga de la definición de las reglas a ser cargadas para evaluarlas en nuestro motor de evaluación estática de código.

2. Subsistemas



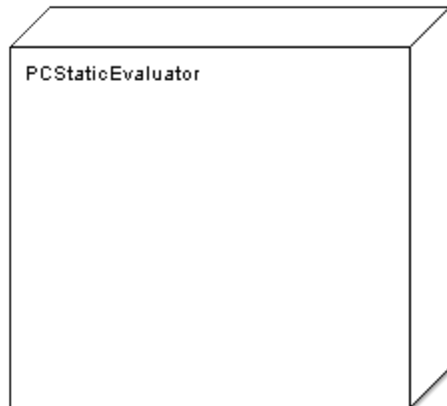
Curso: Tecnología de Objetos	Versión: 1.0
J-Static-Analizer	Fecha: 28/11/2008
Documento de Arquitectura de Software	
http://j-static-analizer.blogspot.com/	

8. Vista de Despliegue

1. Introducción

En esta sección se detalla las configuraciones físicas sobre las cuales se realiza el despliegue del plugin de eclipse. Para el caso del framework JstaticAnalizer se describe como escenario general una distribución esperada para el plugin de eclipse el cual se espera tenga las características descritas anteriormente.

2. Diagramas De Despliegue

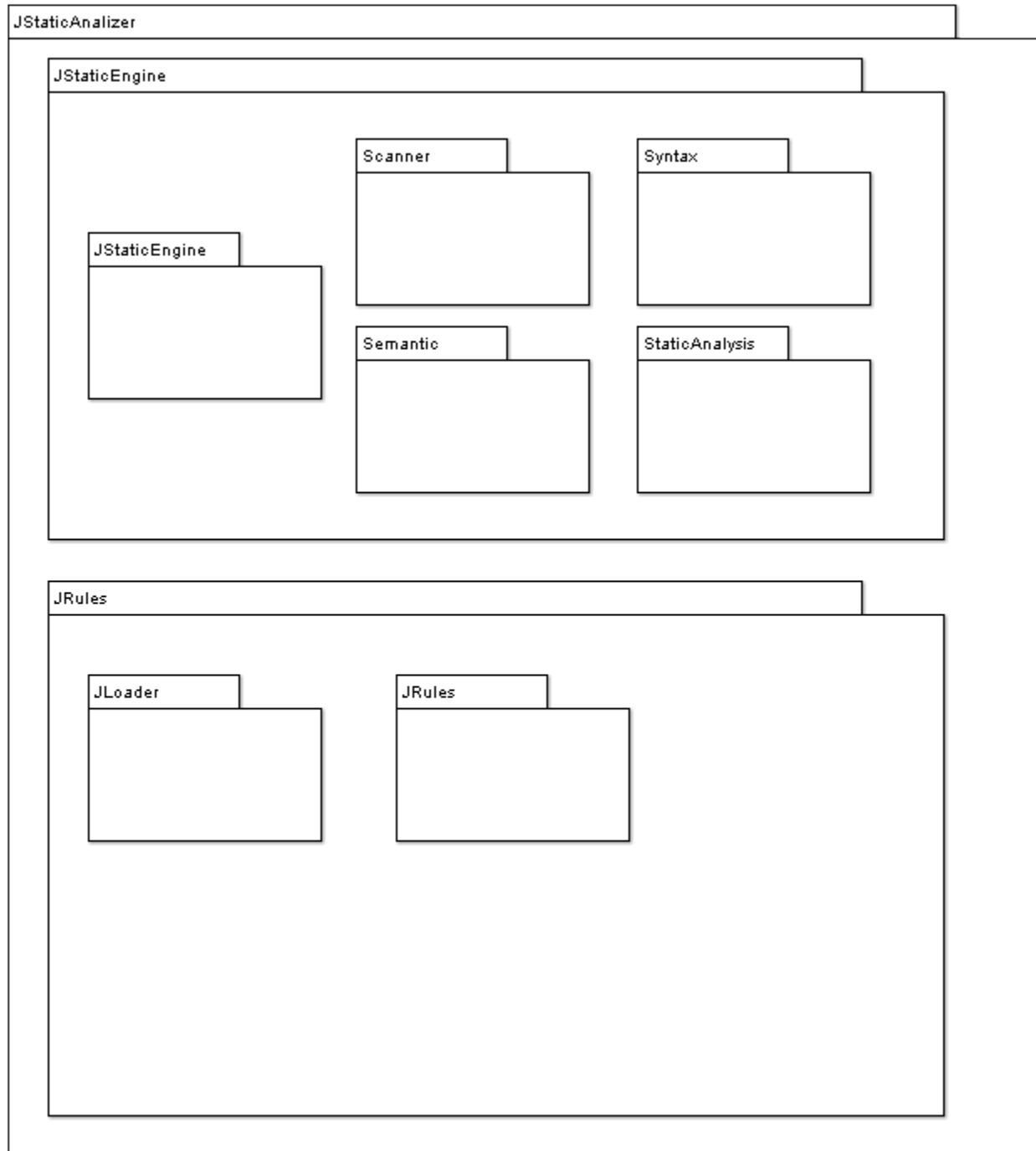


3. PCStaticEvaluator

Este sería el sistema del usuario sobre el cual se ejecuta nuestro plugin para poder hacer la evaluación estática del código java.

Curso: Tecnología de Objetos	Versión: 1.0
J-Static-Analizer	Fecha: 28/11/2008
Documento de Arquitectura de Software	
http://j-static-analizer.blogspot.com/	

4. Arquitectura del Proyecto



Curso: Tecnología de Objetos	Versión: 1.0
J-Static-Analizer	Fecha: 28/11/2008
Documento de Arquitectura de Software	
http://j-static-analizer.blogspot.com/	

➤ **JStaticAnalizer**

Sistema principal del cual se pueden desplegar todos los demás subsistemas para elaborar el proceso de carga de análisis y verificación estática de código java.

➤ **JStaticEngine**

Es el subsistema principal encargado de fungir como sistema de motor de evaluación de nuestro analizador estático de código java.

➤ **JRules**

Reglas del sistema a ser evaluadas por el sistema para poder determinar la existencia o no de falencias en el código escrito por un usuario en común.