**https://lh5.googleusercontent.com/1LuoRHHiiIg3IjmoP_3JO3eY56sySfZVSGCQdO0GFHbWU63bGEzL3ur5ASLLaxt8pAUgswkRwVG7AUMsMPuWdCb3jvMRM6RMuixVKQJeqbw817vK8BDPWpCeSgt0bvpfAu6-NKWFjbqVC9ymhQ**

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores

[CE-1103] Algoritmos y Estructuras de Datos I

Proyecto # 2

Elaborado por:

Francisco J. Murillo Morgan 2015147156

Karina

II Semestre 2017

**Índice**

**Introducción**

**Descripción del Problema**

**Diagramas de clase**

**Descripción de las bibliotecas utilizadas**

**Descripción de las estructuras de datos utilizadas**

**Problemas Encontrados**

**Bitácora de Trabajo**

**Conclusiones**

**Bibliografía**

**Anexos**

**Introducción**

En este proyecto #2 (Code-Insider bautizado por los desarrolladores) se busca realizar un plugin para el interpreté Eclipse, el cual se encargue de brindar información básica del código en ejecución. Con información básica se refiere a un esquema del código completo, cuyas instrucciones tendrán una imagen específica, la cual se puede observar en la figura 1 en los anexos. El objetivo de este proyecto es construir una herramienta la cual facilite el aprendizaje de la programación para estudiantes primerizos, al mostrar la ejecución del código paso por paso con un diagrama de ejecución que se especificará más adelante.

**Descripción del Problema**

Para el plugin Code-Insider, los estudiantes deberán crear un código, el cual se encargue de leer un archivo .java y generar un esquema de ejecución para cada instrucción dentro del código y también se deberá calcular la eficiencia del código analizado. Además de esto, el plugin deberá tener un botón de Step Into y de un Step Over. El Step Over se encargará de brindar la opción de ejecutar el código paso por paso e ira de la mano con el esquema, mostrando la imagen de la instrucción que se encuentra en ejecución. Para el Step Into el programa desplegara el esquema del código del método externo que se desea ver. Por último, el plugin deberá de mostrar los valores de ejecución de cada una de las variables declaradas en el código.

**Diagramas de clase**

Los diagramas de clase para la solución brindada se pueden observar en la sección de anexos, desde la figura 2 en adelante.

**Descripción de las bibliotecas utilizadas**

* Core de eclipse (org.eclipse.core): Esta es la biblioteca principal de Eclipse. En esta se encuentran todas las APIs básicas para poder correr eclipse sin la necesidad de una interfaz de usuario.
* JDT (org.eclipse.jdt): JDT implementa el IDE de Java en eclipse lo cual permite el desarrollo cualquier aplicación Java. Esta librería contiene plugins con los cuales se puede parsear código, debuguear código, y una pequeña UI para explorar paquetes, jerarquías, y ayudantes para crear elementos de java.
* SWT (org.eclipse.swt): Esta biblioteca contiene viñetas gráficas creadas para conformar una interfaz gráfica sencilla de utilizar y agradable para la vista.

**Descripción de las estructuras de datos utilizadas**

Para la implementación de este proyecto, los estudiantes realizaron las siguientes estructuras de datos:

* Node: Este tipo de dato es utilizado para almacenar datos en pequeños paquetes los cuales estarán contenidos en la clase SimpleList, para el almacenamiento de datos en memoria dinámicamente.
* SimpleList: Este tipo de dato es utilizado para almacenar nodos conteniendo diferentes tipos de dato.
* SimpleStatement: Este tipo de dato será utilizado para almacenar la información de una expresión simple de código. Para crear una instancia de SimpleStatement se debe de brindar en el constructor el tipo de expresión, la expresión que se desea almacenar, y la imagen de identificación que representará a esta clase en el esquema. Este tipo de dato tendrá tres atributos (tipo, expresión e imagen) los cuales almacenaran su información respectiva. Este tipo de dato no podrá ser editado una vez creado, lo cual significa que este no posee métodos set, solo tendrá sus respectivos get para obtener el dato contenido en cada uno de sus atributos.
* ConditionStatement: Este tipo de dato hereda todos los métodos y atributos de su clase padre SimpleStatement. Además de tener los atributos de su padre, también contendrá un cuarto atributo que será la condición que debe cumplirse para que la expresión se cumpla. Este tipo de dato es utilizado para representar a todos los if, for, y while que se puedan encontrar en el código que se desea escanear.
* CodeVisitor: Para poder realizar el análisis del código, se esta utilizando una de las API contenida en el JDT llamada AST (Java Abstract Syntax tree). Esta API posee su propio “parser” de código, el cual se encarga de tomar un String que contenga sintaxis de Java y convertirlo en una unidad de compilación, la cual esta conformada por nodos AST (ASTNode) que cada uno de estos representa y contiene una operación a realizar de acuerdo con el código del String pareseado. Para poder recorrer los ASTNode, se requiere de un patrón de diseño Iterator (para el caso de AST se le conoce como visitor) el cual sobre escribe las acciones a realizar dependiendo del tipo de dato que se le pase por parámetro. Este tipo de dato se encargará de contener una versión personalizada del visitor, además de guardar un String que contendrá la complejidad algorítmica calculada para el código escaneado, y una SimpleList que contendrá los elementos ya convertidos para ser utilizados en el esquema.
* CodeScanner: Este tipo de dato será el encargado de realizar los llamados, y creación de instancias, del resto de clases descritas anteriormente. A continuación, se brindará una breve descripción de cada método que posee esta clase.
  + - Constructor que se encargará adjuntar el display a utilizar (necesario a la hora de cargar imágenes en SWT), inicializar la lista de datos para el esquema, e inicializar el String que contendrá la complejidad algorítmica.
    - scanWorkspace(): Este método se encargará de realizar un escaneo del workspace definido de eclipse, y revisara uno a uno los proyectos existentes para cargar sólo el proyecto que se desea escanear. Este método hace el llamado a printProyectInfo().
    - printProyectInfo(): Este método recibe un objeto Iproject el cual contiene el proyecto encontrado que se desea escanear. Este método filtra entre todos los proyectos encontrados, y si este es un proyecto con naturaleza Java, entonces este hará el llamado a printPackageInfos().
    - printPackageInfos(): Este método se encargará de recorrer cada paquete del proyecto, y realizar el escaneo de todos los archivos java que se encuentren en dicho paquete utilizando el método de printICompilationUnitInfo().
    - printICompilationUnitInfo(): Este método tomará los compilation unit (los archivos .java) que se encuentren en el paquete y procederá a realizar el escaneo de cada uno de los archivos. Por cada unidad de compilación que encuentre, esta comenzará a realizar el diagrama correspondiente.
    - printCompilationUnitDetails(): Este método se encarga de tomar el .java y convertirlo en un string, para luego parsearlo, convertirlo en una nueva unidad de compilación, y recórreerlo con el visitor para así obtener todas las declaraciones que este posea y generar el esquema de este código.
* StatementFactory: Esta clase está basada en el patrón de diseñi Abstract Factory. Esta clase contiene dos constructores, uno creará una instancia de SimpleStatement, mientras que el otro creará una instancia de ConditionStatement. La diferencia entre los dos constructores es que el constructor para ConditionStatement recibe un parámetro extra siendo este la condición que activara el ciclo while, for o el condicional if.

**Problemas Encontrados**

**Bitácora de Trabajo**

|  |  |
| --- | --- |
| **Actividad** | **Tiempo** |
| Análisis de requerimientos | 2 horas |
| Diseño de la aplicación | 5 horas |
| Programación | 17 horas |
| Documentación interna | 2 horas |
| Documentación de usuario | 3 horas |
| Documentación técnica | 2 horas |

**Conclusiones**

* Se aprendió un poco más a fondo como funciona internamente un intérprete de código.
* Se adquirió conocimiento acerca de los pasos que debe seguir un debugguer con el fin de brindar todas las características de este.
* Se aprendió como expandir las funciones del intérprete de eclipse a través del uso de plugins.
* Se aprendió un poco más acerca del desarrollo de proyectos open source.

**Bibliografía**

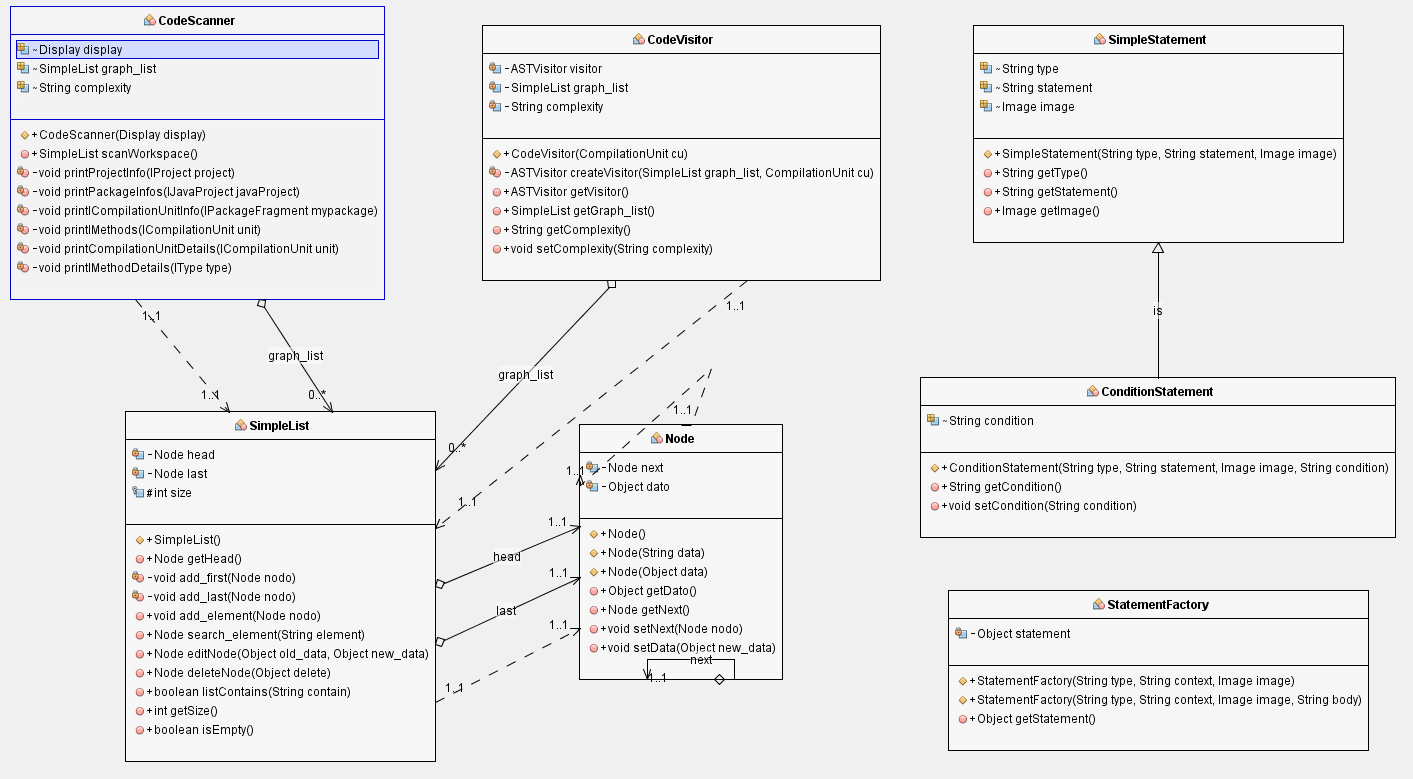
GMBH, V. (2016). *Vogella*. Obtenido de Eclipse IDE Plug-in Development: http://www.vogella.com/tutorials/EclipsePlugin/article.html#download-the-eclipse-sdk

Lars Vogel, S. S. (2009). *Vogella*. Obtenido de Eclipse JDT Abstract Syntax tree and the java model: http://www.vogella.com/tutorials/EclipseJDT/article.html#using-extension-points-to-define-jdt-templates

*progamcreek*. (2015). Obtenido de Use JDT ASTParser to parse single .java: https://www.programcreek.com/2011/11/use-jdt-astparser-to-parse-java-file/

Redko, A. (Septiembre de 2013). *Oracle Docs*. Obtenido de {http://docs.oracle.com/javafx/2/ui\_controls/jfxpub-ui\_controls.html

**Anexos**



**Figura #2 Diagrama de clases.**