CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de

Ingenieros Informáticos

TRABAJO FIN DE GRADO

AED-Viewer:
Visualizador de Estructuras de Datos en Java

Autor: Mario Martín Ricote

Director: Guillermo Román

MADRID, JUNIO 2017

Índice

Índice de Imágenes

AGRADECIMIENTOS

Antes de sumergirnos en el proyecto de fin de grado me gustaría dedicar unas primeras palabras para aquellos que me ayudaron en este camino, no solo en el TFG si no a lo largo de la carrera, porque a veces el camino puede ser duro y tengas ganas de abandonar, pero hay personas que te hacen seguir luchando por lo que quieres por muchas dificultades que tengas en el camino.

Para empezar, me gustaría agradecer a mi tutor de este trabajo, Guillermo Román porque se ha esforzado en ayudarme a que este trabajo saliera adelante y por estar siempre cuando le necesitaba. Además, tengo un buen recuerdo de él como profesor por tanto creo que es una de las personas idóneas para compartir este trabajo.

A mis padres el esfuerzo que han hecho económicamente para que yo estudiara lo que me gustaba y haber cumplido un sueño.

A mi familia por apoyarme siempre cuando lo he necesitado.

Por últimos a unos compañeros y amigos que me llevo de esta facultad y que espero recorrer este camino de la vida con ellos. Cosmin, Michael y Jesús que me han ayudado en este largo camino y han tenido mucha paciencia conmigo para hacerme ver que cualquier cosa se podía conseguir.

RESUMEN

Uno de los aspectos clave en el aprendizaje la de programación es la comprensión de la estructura interna de los datos utilizados en el programa. El objetivo del presente trabajo es ofrecer una herramienta a los estudiantes de las diferentes asignaturas de programación que les permita visualizar los Tipos Abstractos de Datos con los que trabajan, con el objetivo de ayudarles a comprender mejor el funcionamiento de cada estructura, así como las diferencias entre unas estructuras y otras. Para ello se desarrollará una herramienta (plugin) que mostrará una representación visual de las diferentes estructuras y que se integrará con el depurador de en un entorno de desarrollo profesional, de forma que éste pueda ser utilizado por los estudiantes durante la realización de las prácticas de las asignaturas de programación.

El presente trabajo muestra una visión general de los Tipos Abstractos de Datos más comúnmente estudiados en las asignaturas de programación, entre los que podemos hablar de listas, pilas, colas, árboles, …, y una representación visual de los mismos. El lenguaje de programación elegido para la visualización de las estructuras es Java, y el entorno de desarrollo elegido para la integración de la herramienta ha sido Eclipse.

ABSTRACT

One key factor in learning computer programming is the understanding of internal structures of the data used in the computer program. The objective of this work is offer a tool to the students of the different programming subjects that it allows them to visualize the Abstract Data Types with which they work, with the objective of helping them to better understanding the features of each structure, as well as the difference between some structures. In order to fulfill this objective a tool (plugin) will be developed that will show a visual representation of the different structures and will be integrated with the debugger in a professional development environment, so that it would be used for the students while the execution of the practices of programming subjects.

The present work exposed a general view of the Abstract Data Types most commonly studies in the programming subject, among this types we can speak of lists, stacks, tails, trees …, and a visual representation them. The programming language chosen for the visualization of the structures in Java and the development environment chosen for the integration of the tool has been Eclipse.

1 INTRODUCCIÓN

Una estructura de datos es un conjunto de datos y una serie de funciones que representan entidades, permitiendo separar, agrupar y organizar los datos, así como sus funcionalidades. El presente trabajo de fin de grado consiste en el desarrollo de una herramienta que permita mostrar una representación visual de los Tipos Abstractos de Datos más comunes en programación. En las asignaturas de enseñanza de programación es frecuente encontrar alumnos con dificultades a la hora de comprender tanto la representación como el funcionamiento interno de las operaciones disponibles para las estructuras de datos estudiadas en la asignatura. El objetivo principal del presente proyecto es ayudar a los estudiantes de programación a entender cómo funcionan dichas estructuras de datos mediante la visualización del contenido de la estructura en tiempo de ejecución. De este modo se pretende facilitar el aprendizaje y el rendimiento de los estudiantes de las asignaturas de programación.

Las estructuras de datos y los datos contenidos en ellas van cambiando durante la ejecución de un programa y poder visualizar los cambios en tiempo de ejecución es fundamental para que la herramienta pueda ser de utilidad para los estudiantes. Asimismo, para que dicha herramienta pueda ser de utilidad, debe poder ser utilizada durante el desarrollo, es decir, debe integrarse en algún entorno de desarrollo (IDE) que los estudiantes utilicen para hacer sus programas. Los entornos de desarrollo actuales permiten la incorporación de nuevas funcionalidades y herramientas (plugins). De la misma forma, los IDE's permiten también el acceso a la información contenida en los diferentes componentes que conforman el IDE. En este caso, al estar interesados en obtener la información en tiempo de ejecución, el plugin desarrollado debe integrarse con el depurador del IDE, y de esta forma extraer la información acerca de la estructura y los datos contenidos en la misma en tiempo de ejecución. Después de llevar a cabo un estudio de los diferentes entornos, Eclipse, NetBeans, IntelliJ Idea; al ser Eclipse el entorno más utilizado por los alumnos y que presenta unas posibilidades de extensión mayores, el trabajo se centrará en el desarrollo de un plugin para el IDE Eclipse.

Debido al enorme número de posibles TAD's y el elevado número de implementaciones que se pueden encontrar, al tener como objetivo la formación de los alumnos de las asignaturas de programación de la E.T.S. de Ingenieros Informáticos de la Universidad Politécnica de Madrid el trabajo se centrará en las estructuras de datos estudiadas en la asignatura *Algoritmos y estructuras de datos* (AED), que se cursa en el tercer semestre, tanto del Grado en Ingeniería Informática, como en el Grado en Matemáticas e Informática. En la asignatura anteriormente citada se estudian las estructuras de: listas, colas FIFO/LIFO, colas con prioridad, árboles y HashMap. Dichas estructuras se encuentran implementadas en Java, lenguaje que se integra perfectamente con el entorno de desarrollo elegido para el proyecto.

Los objetivos del presente proyecto los siguientes:

- Estudio del arte de librerías graficas de Java para representar visualmente las estructuras de datos en una vista de Eclipse mediante la implementación de un plugin para el IDE
- Diseñar la representación visual de los Tipos Abstractos de Datos impartidas en la asignatura Algoritmo y Estructura de Datos
- Implementar la visualización de las estructuras de datos en una vista creada para Eclipse
- Integrar mediante un plugin la representación visual de las estructuras en Eclipse
- Integrar botones para el pintado y borrado de las estructuras en la vista
- Representar los datos de las variables contenidas dentro de cada objeto que compone la estructura

2 TIPOS ABSTRACTOS DE DATOS

Para comenzar hablaremos sobre los Tipos Abstractos de Datos impartidas en la asignatura Algoritmos y Estructura de datos y como es su visualización en Eclipse.

2.1 Listas de Posiciones

Una lista representa una secuencia de n objetos ordenados de forma lineal tal que cada elemento está en una posición llamado índice. Cada índice del elemento se calcula contando los elementos que tenga anteriores a este.

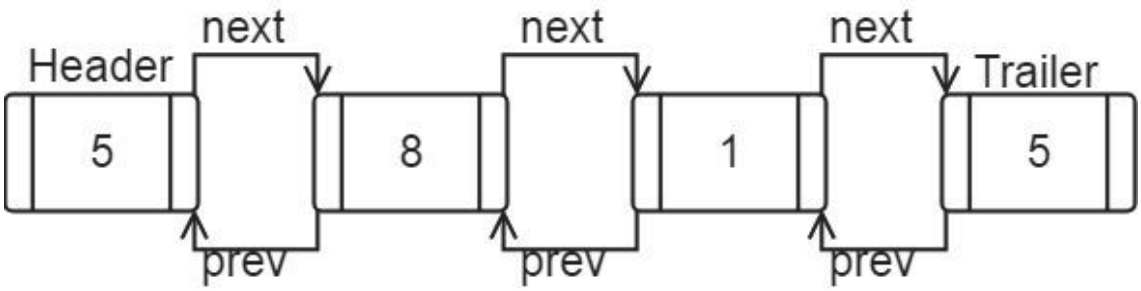

Ilustración 1 Ejemplo de lista

En eclipse se representa mediante rectángulos unidos por flechas tanto al rectángulo de la posición izquierda como el de la derecha si los tuviera. Esto se debe a que las listas estan enlazadas tanto con su elemento anterior como su posterior. La clase que usamos para las listas es PositionList. Otra característica de este tipo de listas es que sólo eliminamos una posición las posiciones que estén enlazadas con el elemento eliminado se unen entre ellas. Por tanto no afectaría el recorrido entre posiciones.

Interfaz PositionList

```
package aed.positionlist;

/**
```

```
 * A reimplementation of PositionList.java from the net.datastructures
library.
 * @author aed
 *
 * @param <E> The type of the element contained in the Position
 */
public interface PositionList<E> extends Iterable<E> {

        /**
         * Returns the number of nodes in the list
         * @return Returns the number of nodes in the list
         */
        public int size();


        /**
         * Returns true if the list is empty (has no nodes), false if it's
not.
         * @return Returns true if the list is empty (has no nodes), false if
it's not.
         */
        public boolean isEmpty();


        /**
         * Returns the first node of the list, or null if the list is empty.
         * @return Returns the first node of the list, or null if the list is
empty.
        */
        public Position<E> first();


        /**
         * Returns the last node of the list, or null if the list is empty
         * @return Returns the last node of the list, or null if the list is
empty
         */
        public Position<E> last();



        /**
         * Either returns the next node to the right of parameter node 'p' in
the
         *  list, or returns null if 'p' doesn't have a next node to the
right.
         * @param p Position to get it's next node
         * @return returns the next node to the right of parameter node 'p'
in the
         *  list, or returns null if 'p' doesn't have a next node to the
right
         * @throws IllegalArgumentException Raises the exception if 'p' is
null or is not a node of the list.
         */
        public Position<E> next(Position<E> p) throws
IllegalArgumentException;


        /** Either returns the previous node to the left of parameter node
'p' in
         *  the list, or returns null if 'p' doesn't have a previous node to
the
         *  left
```

```
         * @param p Position to get it's previous node
         * @return returns the previous node to the left of parameter node
'p' in
         *  the list, or returns null if 'p' doesn't have a previous node to
the
         *  left
         * @throws IllegalArgumentException Raises the exception if p is null
or is not a node of the list.
         */
        public Position<E> prev(Position<E> p) throws
IllegalArgumentException;


        /**
         * Inserts a new first node to the list with 'elem' as element
         * @param elem The element to be inserted
         */
        public void addFirst(E elem);


        /**
         * Inserts a new last node to the list with 'elem' as element
         * @param elem The element to be inserted
         */
        public void addLast(E elem);


        /**
         * Inserts a new node with 'elem' as element to the list right before
         * parameter node 'p'.
         * @param p The node where the element is inserted before
         * @param elem The element to be added
         * @throws IllegalArgumentException Raises the exception if 'p' is
null or is not a node
         *  of the list.
         */
        public void addBefore(Position<E> p, E elem) throws
IllegalArgumentException;


        /**
         * Inserts a new node with 'elem' as element to the list right after
         * parameter node 'p'
         * @param p The node where the element is inserted after
         * @param elem The element to be added
         * @throws IllegalArgumentException Raises the exception if 'p' is
null or is not a node
         *  of the list.
         */
        public void addAfter(Position<E> p, E elem) throws
IllegalArgumentException;


        /**
         * Removes node 'p' from the list and returns its element.
         * @param p The position to be removed
         * @return The element that has been removed
         * @throws IllegalArgumentException Raises the
         *  exception if 'p' is null or is not a node of the list.
         */
        public E remove(Position<E> p) throws IllegalArgumentException;
```

```
        /**
         * Sets the element of node 'p' on the list to 'elem' and returns
         * @param p The position where the element is et
         * @param elem The element contained in p before the update
         * @return the old element in 'p'
         * @throws IllegalArgumentException Raises the exception if 'p' is
null or is not a node of
         *  the list.
         */
        public E set(Position<E> p, E elem) throws IllegalArgumentException;


        /**
         * Returns an array with all the elements of the list or an empty
array
         *  of length zero if the list is empty.
         * @return Returns an array with all the elements of the list or an
empty array
         *  of length zero if the list is empty.
         */
        public Object [] toArray();
}
```

Ejemplo de lista

```
//Creamos una lista
PositionList<String> lista2 = new NodePositionList<>();

//Añadimos elementos a la lista
for(int i=0;i<3;i++){
lista2.addLast("Elemento "+i);
}

//Comprobamos el tamaño de la lista con size() si el tamaño es distinto a
//6 añadimos en la primera posicion un elemento
if(lista2.size()!=4){
lista2.addFirst("El primero");
}

//Sacamos el primer elemento de la lista con Position
aed.positionlist.Position<String> pos=lista2.first();
//Mostramos cada elemento y recorremos la lista con next
for(int i=0;i<lista2.size();i++){
System.out.println(pos.element());
pos=lista2.next(pos);
}
//Cogemos el ultimo elemento y reccoremos la lista hacia atras
pos=lista2.last();
for(int i=0;i<lista2.size();i++){
System.out.println(pos.element()+" al reves");
pos=lista2.prev(pos);
}
```

```
El primero
Elemento 0
Elemento 1
Elemento 2
Elemento 2 al reves
Elemento 1 al reves
Elemento 0 al reves
El primero al reves
```

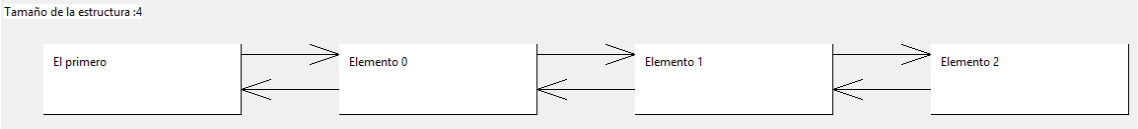

Ilustración 2 Ejemplo de lista desde el plugin

2.2 FIFO

First In First Out (FIFO) representa una cola con orden. Cada vez que un elemento se añade a la cola este se pone al final de esta, si se desea sacar un elemento se sacará el primero que entro desplazando una posición todos los elementos que estén detrás de él.

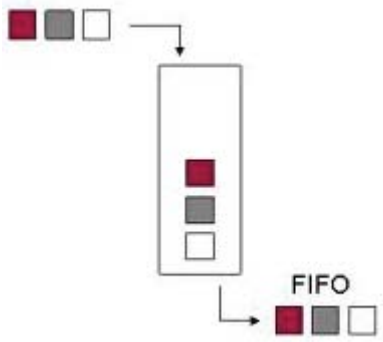

Ilustración 3 FIFO

En la representación en Eclipse se ha optado por un contenedor en el que van acumulándose los objetos por orden descendente. Se puede comprobar que si el tamaño de la cola fuera más de un determinado número de elementos solo aparecerían los 6 primeros ya que en la cola FIFO como hemos mencionado anteriormente lo que puede ser útil para entender FIFO son los elementos que se encuentran en las primeras posiciones.

Interfaz FIFO

```
package aed.fifo;

import java.lang.Iterable;

import aed.positionlist.*;

/**
 * Interface for a FIFO TAD
 * @author aed
 *
 * @param <E> The type of the element contained in the FIFO
 */

public interface FIFO<E> extends Iterable<E> {

        /**
         * Returns the number of elements in the FIFO
         * @return The number of elements in the FIFO
```

```
         */
        public int size();

        /**
         *  Returns true if the FIFO is empty
         * @return true if the FIFO is empty
         */
        public boolean isEmpty();

        /**
         * Returns the first element in the FIFO if non-empty
         * @return The first element in the FIFO if non-empty
         * @throws EmptyFIFOException If the FIFO is empty
         */
        public E first() throws EmptyFIFOException;


        /**
         * Enqueues the element as the last element of FIFO.
         * @param elem The element to be enqueued
         */     public void enqueue(E elem);

        /**
         * Dequeues the first element in the FIFO if non-empty
         * @throws EmptyFIFOException If the FIFO is empty
         */
        public void dequeue() throws EmptyFIFOException;

        /**
         * Returns an array with all the elements of the FIFO or an empty
array
         *  of length zero if the FIFO is empty.
         * @return An array with all the elements of the FIFO or an empty
array
         *  of length zero if the FIFO is empty.
         */
        public Object [] toArray();

        /**
         *  Returns a list with all the elements of the FIFO or an empty list
if
         *  the FIFO is empty.
         * @return a list with all the elements of the FIFO or an empty list
if
         *  the FIFO is empty
         */
        public PositionList<E> toPositionList();
}
```

```
//Creamos la cola FIFO
FIFO<String> fifo = new FIFOList<String>();

//Comprobamos que la cola esta vacia y si es asi insertamos un elemento
if(fifo.isEmpty()){
fifo.enqueue("Primero");
}
//Insertamos varios elementos
for(int i=0;i<5;i++){
fifo.enqueue("Elemento "+i);
}
//Sacamos el tamaño de la lista
System.out.println("Tamaño :"+fifo.size());
```

```
//Imprimimos los valores a medida que sacamos elementos de la cola
for(int i=0;i<fifo.size();i++){
System.out.println(fifo.first());
fifo.dequeue();
}
```

```
Primero
Elemento 0
Elemento 1
Elemento 2
Elemento 3
Elemento 4
```

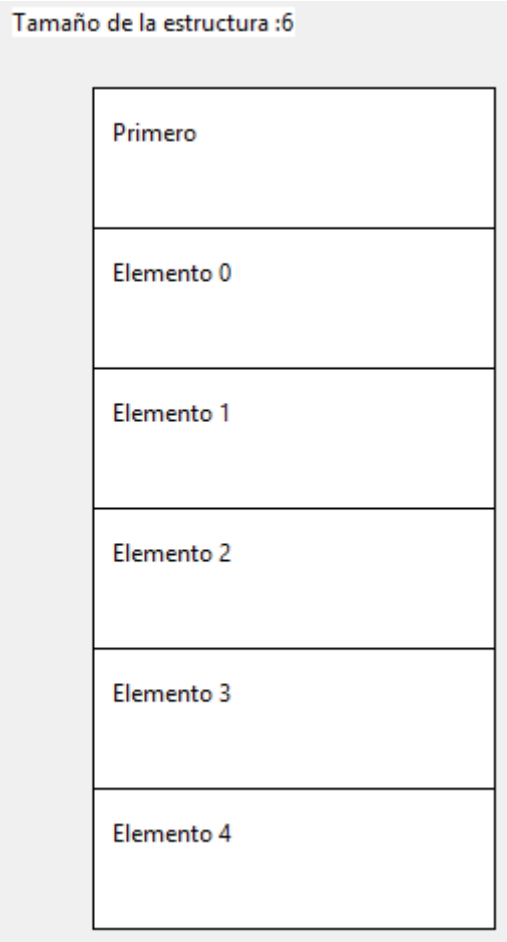

Ilustración 4 Ejemplo de FIFO desde el plugin

2.3 LIFO

“Last In Last Out, es una cola con orden. Al añadir un elemento se pone detrás de los elementos ya introducidos, si se quiere sacar un elemento comenzara a sacar el elemento de la última posición que este en la cola.

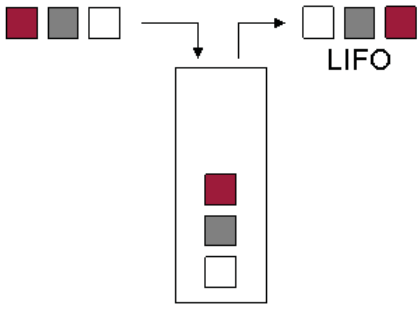

Ilustración 5 LIFO

Para LIFO como se puede observar en la figura siguiente representa un contenedor en el que se almacenan en orden ascendente los elementos, pero en la visualización de la vista tenemos la visualización de los primeros y últimos elementos de la cola, ya que son, los que se podrían considerar interesantes para observar el comportamiento de la cola al insertar y sacar elemento de una LIFO.

Interfaz LIFO

```
package aed.lifo;

import java.util.EmptyStackException;
import aed.positionlist.*;
import java.lang.Iterable;

/**
 * The LIFO inteface represents a last-in-first-out (LIFO) stack of objects
 * @author aed
 *
 * @param <E> The type of the elements contained in the stack
 */
public interface LIFO<E> extends Iterable<E> {

        /**
         * Returns the number of elements
         * @return The number of elements
         */
        public int size();

        /** */
        /**
         * Returns true if the stack is empty
         * @return Returns true if the stack is empty
         */
        public boolean isEmpty();
```

```
        /**
         * Returns the element on top of the LIFO if non-empty
         * @return the element on top of the LIFO if non-empty
         * @throws EmptyStackException if the stack is empty
         */
        public E top() throws EmptyStackException;

        /**
         * Removes the element on top of the LIFO if non-empty
         * @throws EmptyStackException if the stack is empty
         */
        public void pop() throws EmptyStackException;

        /**
         * Pushes the element e on top of the LIFO.
         * @param elem The element to be pushed in the stack
         */
        public void push(E elem);

        /**
         *  Returns an array with all the elements of the LIFO or an empty
array
         *  of length zero if the LIFO is empty.
         * @return An array with all the elements of the LIFO or an empty
array
         *  of length zero if the LIFO is empty.
         */
        public Object [] toArray();

        /**
         *  Returns a list with all the elements of the LIFO or an empty list
if
         *  the LIFO is empty.
         * @return A list with all the elements of the LIFO or an empty list
if
         *  the LIFO is empty
         */
        public PositionList<E> toPositionList();
}
```

```
//Creamos la cola LIFO
                    LIFO<String> lifo = new LIFOList<String>();
                    //Comprobamos que la cola esta vacia y si es asi
insertamos un elemento
                    if(lifo.isEmpty()){
                          lifo.push("Primero");
                    }
                    //Insertamos varios elementos
                    for(int i=0;i<4;i++){
                          lifo.push("Elemento :"+i);
                    }
                    //Sacamos el tamaño de la lista
                    int tam=lifo.size();

                    //Imprimimos los valores a medida que sacamos elementos de
la cola
                    for(int i=0;i<tam;i++){
                          System.out.println(lifo.top());
                          lifo.pop();
```

```
                    }
Elemento :3
Elemento :2
Elemento :1
Elemento :0
Primero
```

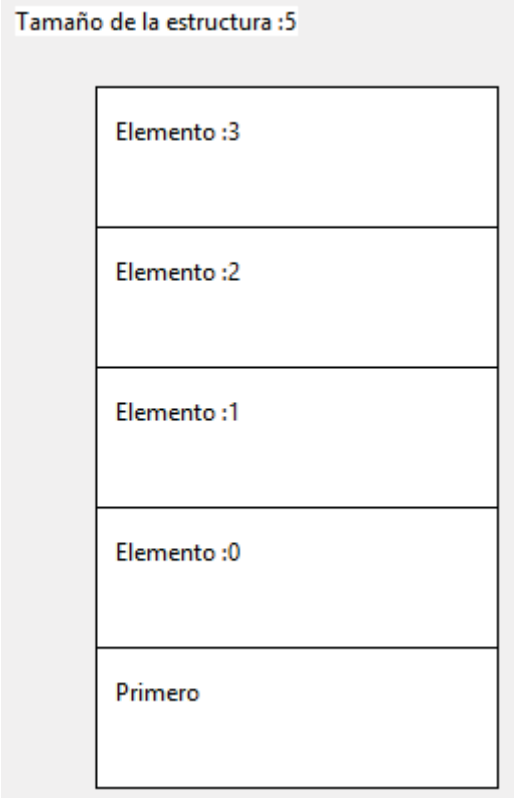

Ilustración 6 Ejemplo de LIFO desde el plugin

Un ejemplo para comparar las colas FIFO y LIFO en el que podemos observar que la inserción de elementos es la misma. Por el contrario, las extracciones de elementos se hacen de forma diferente.

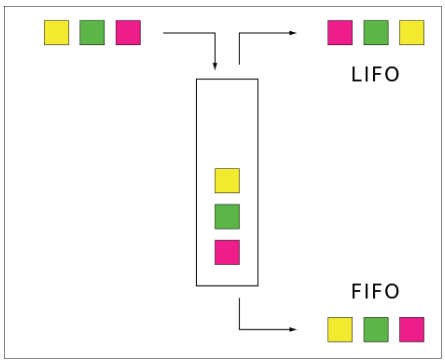

Ilustración 7 Comparación de FIFO y LIFO

2.4 Colas con Prioridad

Las colas con prioridad (priority queues) consisten en una colección de elementos llamados valores a los que se les asignan una clave por cada valor. A diferencia de las colas, que el orden de salida depende directamente del orden de llegada, en el caso de las colas con prioridad, el orden de salida depende la prioridad de la clave de la entrada. En general, y también en las colas con prioridad de la asignatura, los elementos que saldrán primero son aquellos con mayor prioridad, que con aquellos con una clave de menor valor.

Interfaz PriorityQueue

```
package net.datastructures;

//begin#fragment PriorityQueue
/** Interface for the priority queue ADT */
public interface PriorityQueue<K,V> {
  /** Returns the number of items in the priority queue. */
  public int size();
  /** Returns whether the priority queue is empty. */
  public boolean isEmpty();
  /** Returns but does not remove an entry with minimum key. */
  public Entry<K,V> min() throws EmptyPriorityQueueException;
  /** Inserts a key-value pair and return the entry created. */
  public Entry<K,V> insert(K key, V value) throws InvalidKeyException;
  /** Removes and returns an entry with minimum key. */
  public Entry<K,V> removeMin() throws EmptyPriorityQueueException;
}
//end#fragment PriorityQueue
```

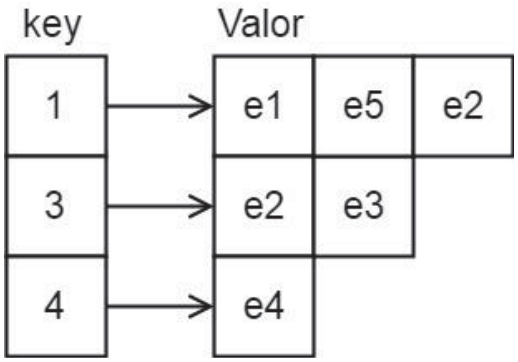

Ilustración 8 Cola con prioridad

```
//Inicializamos la cola y añadimos unos cuantos pares de clave - valor
PriorityQueue<String,String> pq = new SortedListPriorityQueue<>();
pq.insert("1", "Melon");
pq.insert("3", "Verano");
pq.insert("4", "Piscina");
pq.insert("11", "Playa");
pq.insert("12", "Agua");
pq.insert("3", "Vacaciones");

//Recorremos la lista hasta que se vacie y sacamos el valor de mayor prioridad
de la cola
```

```
while(!pq.isEmpty()){
Entry<String, String> val=pq.removeMin();
System.out.println(val.getKey()+" - "+val.getValue());
}
```

```
1 - Melon
11 - Playa
12 - Agua
3 - Vacaciones
3 - Verano
4 - Piscina
```

Para la representación de nuestro plug-in se ha representado de la siguiente manera:

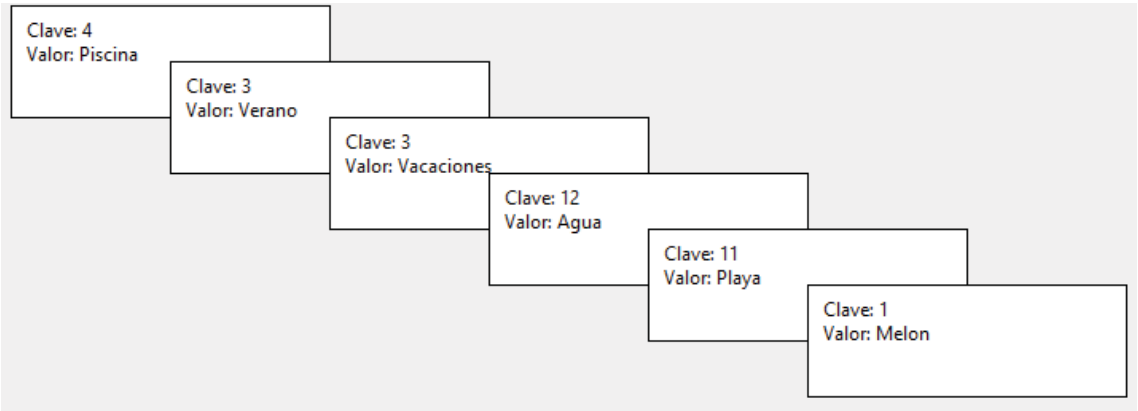

Ilustración 9 Visualización de cola con prioridad en plugin

Como podemos observar los elementos se han colocado de forma alfabética en que el elemento con la clave más prioritaria este encima de las demás. Esta representación sirve para observar el orden de los elementos asi como el orden en el que salen de la cola.

2.5 Mapas (maps)

Un Mapa o Map es una colección de elementos de forma no ordenada, es una estructura que se puede representar como una tabla de dos columnas, una para la clave y otra para el valor. Cada elemento se identifica con una clave única. Estas estructuras son útiles si no necesitas conocer el tamaño de la colección permitiendo identificar el valor en base a la clave. Las claves tiene que ser unicas aunque el valor que las relaciona puede tener el mismo valor.

Interfaz de Map

```
package net.datastructures;
/**
 * An interface for a map which binds a key uniquely to a value.
 * @author Michael Goodrich
 */
//begin#fragment Map
// A simple Map interface
public interface Map<K,V> {
        /** Returns the number of items in the map. */
        public int size();
```

```
        /** Returns whether the map is empty. */
        public boolean isEmpty();
        /** Puts a given key-value pair in the map, replacing a previous
         * one, if any, and returns the old value. */
        public V put(K key, V value) throws InvalidKeyException;
        /** Returns the value associated with a key. */
        public V get(K key) throws InvalidKeyException;
        /** Removes the key-value pair with a given key. */
        public V remove(K key) throws InvalidKeyException;
        /** Returns an iterable object containing all the keys in the map. */
        public Iterable<K> keySet();
        /** Returns an iterable object containing all the values in the map.
*/
        public Iterable<V> values();
        /** Returns an iterable object containing all the entries in the map.
*/
        public Iterable<Entry<K,V>> entrySet();
}
//end#fragment Map
```

```
//Inicializamos el Map
Map<String,String> map = new HashTableMap();
//Añadimos unos cuantos elementos
map.put("0", "Soy el primero");
for(int i=1;i<6;i++){
map.put(""+i+"", "Elemento"+i*1.5);
}
//Sacamaos elementos de la lista
for(int i=0;i<5;i++){
System.out.println(map.remove(""+i+""));
}
```

```
Soy el primero
Elemento1.5
Elemento3.0
Elemento4.5
Elemento6.0
```

En la representación de Eclipse se muestra de la siguiente manera:

Tamaño de la estructura :6

Clave	Valor
1	Elemento1.5
0	Soy el primero
5	Elemento7.5
4	Elemento6.0
3	Elemento4.5
2	Elemento3.0

Ilustración 10 Map

2.6 Tree

Los árboles son estructuras que se componen de nodos de los que derivan o son derivados de otros nodos formando una asociación de padre e hijo entre nodos. Hay un nodo principal que se denomina raíz del que cuelgan sus nodos hijos y no tiene nodo padre. Cada nodo tiene un valor que puede ser el mismo para otro.

Cada descendencia de padre a hijas ocupa un nivel siendo el nivel máximo el nodo que este mas debajo de la estructura y el nivel del árbol será el recorrido desde el nodo raíz al nodo más bajo del árbol.

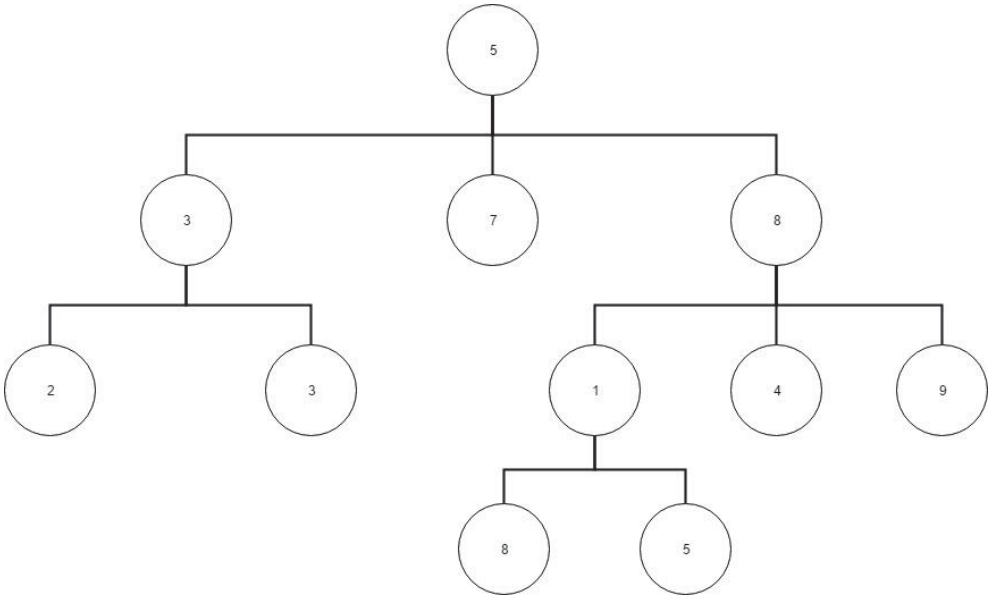

Ilustración 11 Tree

Para mejorar la presentación visual y evitar que el árbol ocupe mucho espacio a lo ancho, otra posible representación de un árbol sería de forma vertical, donde los hijos de cada nodo del árbol se forman hacia abajo y cada hijo se desplaza hacia la derecha.

Ilustración 12 Árbol desde el plugin

3 REQUISITOS

Los requisitos darán a conocer las funcionalidades que se quieren implementar, así como las restricciones que debe tener el sistema.

REQ-01: El sistema creara una vista llamada Visualizador para la visualización de las variables.

REQ-02: El sistema tendrá en el menú de vistas una categoría Visualizador de estructuras y la vista Visualizador para mostrar en la ventana de Eclipse.

REQ-03: El sistema tendrá un botón y una opción en el menú llamados “Visualizador” para dibujar una variable de Tipo Abstracto en la vista “Visualizador”.

REQ-04: El sistema tendrá un botón y una opción en el menú llamados “Limpiar” para eliminar todos los objetos dibujados en la vista “Visualizador”.

REQ-05: El sistema mostrara en una lista todas las variables del Tipo Abstracto que haya leído del depurador.

REQ-06: El sistema comprobara que el depurador este activado, si no es así, mostrara un mensaje de aviso.

REQ-07: El sistema pintara en la vista “Visualizador” la estructura seleccionada del menú mostrado al seleccionar la opción “Visualizar” y el tamaño de la estructura.

REQ-08: El sistema mostrara en la vista el contenido del objeto limitándolo a un número determinado de líneas y un ancho de texto.

REQ-09: El sistema pintará los rectángulos de un mismo tamaño para todos los tipos abstractos que estén implementados.

REQ-10: El sistema mostrara en una ventana todo el contenido del objeto cuando se seleccione uno de los rectángulos.

REQ-11: El sistema colocará adecuadamente el texto para mostrar los atributos de los objetos.

REQ-12: El sistema repintara la estructura si se hace scroll sobre la vista.

REQ-13: El sistema repintara la vista “Visualizador” cuando se seleccione una nueva estructura.

REQ-14: El sistema mostrara las colas FIFO y LIFO como un rectángulo encima de otro.

REQ-15: El sistema mostrara los datos de Tipo Abstracto Map como una tabla.

REQ-16: El sistema mostrará las Colas con Prioridad como una serie de rectángulos uno encima de otro mostrando clave y valor.

REQ-17: El sistema mostrara los objetos de las listas de forma horizontal unidos por flechas.

REQ-18: El sistema mostrara un árbol en el que los hijos de un nodo del árbol se puedan mostrar u ocultar.

4 ENTORNO DE DESARROLLO ECLIPSE

Eclipse (www.eclipse.org) es un entorno integrado de desarrollo (IDE) que incorpora múltiples funcionalidades para el desarrollo de programas. Se trata de un software de código abierto, desarrollado en Java y que se compone de un workbench en que se montan diversas herramientas y complementos. Dichas herramientas y complementos permiten una personalización total del entorno de desarrollo en función de las necesidades de cada proyecto y lenguaje de programación.

La arquitectura de Eclipse también permite integración con otros lenguajes como C, procesadores de texto como LaTeX (https://www.latex-project.org) y aplicaciones de red. También integra frameworks para el desarrollo de aplicaciones gráficas, definición y manipulación de datos como por ejemplo GEF, XML o herramientas UML.

A continuación, se muestran las partes de las que se compone Eclipse:

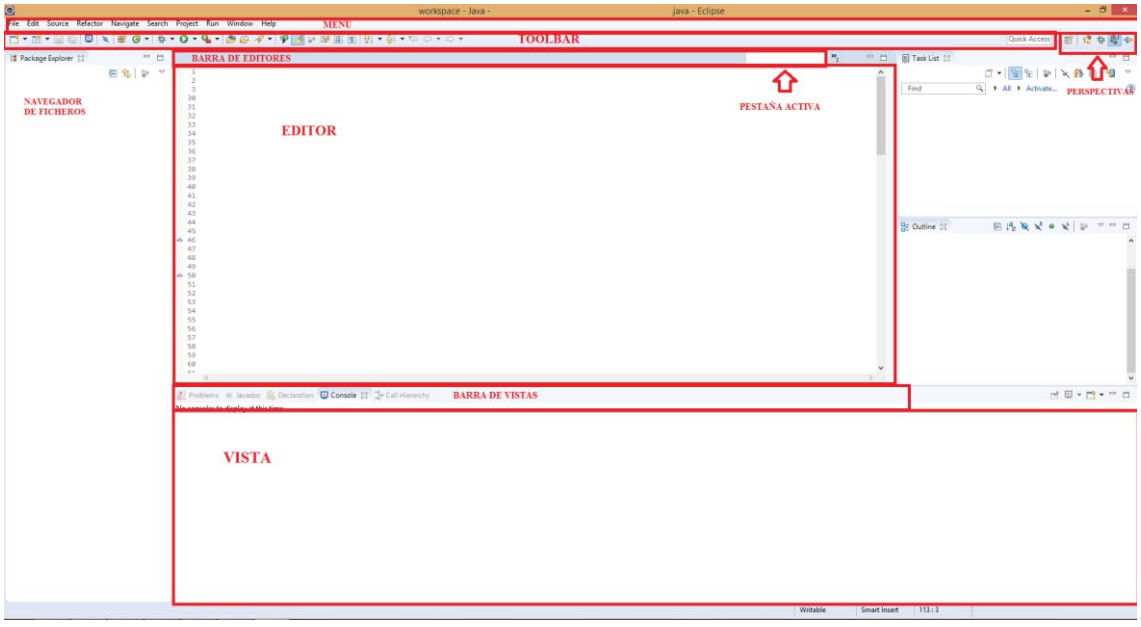

Ilustración 13 Partes de eclipse

Eclipse está formado por perspectivas y estas a su vez la componen vistas y editores. La barra superior del eclipse se le denomina toolbar en el contienen distintos botones que ejecutan comandos, hay algunos comandos que ya aparecen en todas las distribuciones

de Eclipse predefinidos como ejecutar o debug. Las perspectivas puedes personalizarlas con cualquier vista, opción de menú, editores.

La ventana principal es el editor en el que se permite escribir código de programas, el editor permite tener varios editores abiertos que se almacenan en pestañas para poder moverte por los distintos editores.

Las vistas son las ventanas secundarias que implementan componentes para distintas funcionalidades como mostrar variables, ver el directorio de una carpeta. Cada uno de estos componentes puede ser personalizado y extendido para incluir nuevas funcionalidades a las ya predefinidas en el entorno de Eclipse.

4.1 Extensiones del plugin AED-Viewer

De cara a nuestro plugin se implementará una nueva opción en el menú llamado AEDViewer que implementará la función "Visualizar" y "Limpiar". También estas funciones se implementarán en la barra Toolbar con forma de iconos en el que se puede observar la función al pasar por encima del icono. Estas implementaciones se pueden ver en la siguiente figura.

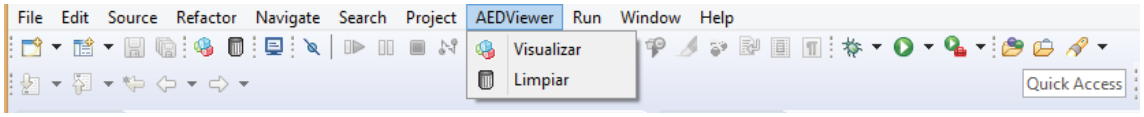

Ilustración 14 Visionado del menú implementado con el plugin

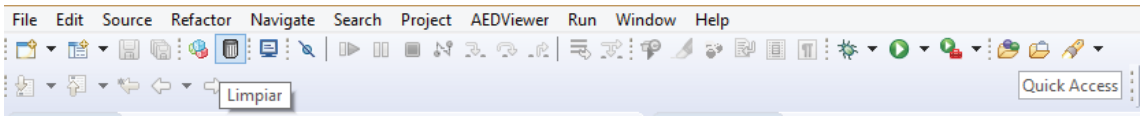

Ilustración 15 Visionado de los nuevos iconos con el plugin

La vista se llama "Visualizador" y tiene el mismo icono que "Visualizar"

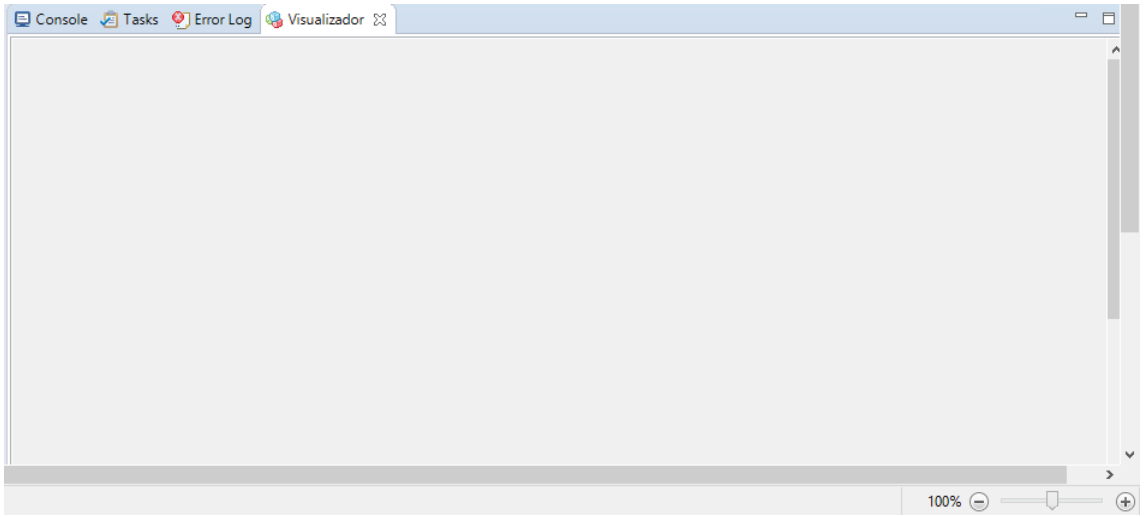

Ilustración 16 Ejemplo del visionado de la vista

Para tener nuestra vista en pantalla tendremos que añadirlo a través del visualizador de vistas.

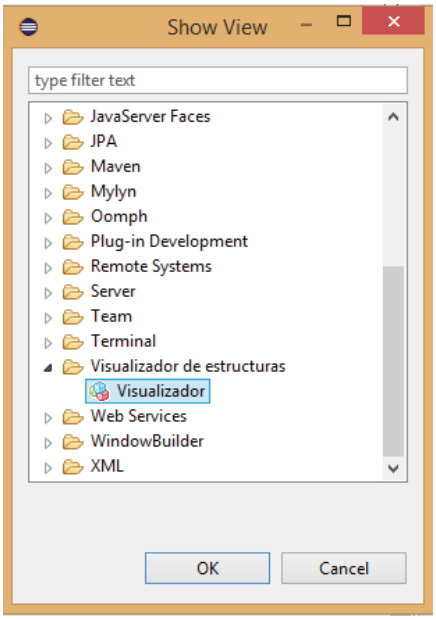

Ilustración 17 Visor de vistas

Para ello se tiene que escribir en plugin.xml los componentes descritos anteriormente. A continuación, se mostrará el código de plugin.xml hay que destacar que algunas funciones de explicaran más adelante en la sección de listado de clases.

Esta extensión sirve para declarar la vista personalizarla y añadirlo al visualizador de estructuras con una categoría en el menú del visualizador de vistas.

```
<extension
      point="org.eclipse.ui.views">
   <category
         name="Visualizador de estructuras"
         id="Visualizador de estructuras">
   </category>
   <view
         name="Visualizador"
         icon="icons/icon_component.gif"
         category="Visualizador de estructuras"
         class="pluginvista.views.Vista"
         id="pluginvista.views.Vista">
   </view>
</extension>
```

Esta extensión sirve añadir en una sección de la perspectiva la vista en nuestro caso se añadirá en la zona de navegación donde también se puede encontrar la consola

```
<extension
      point="org.eclipse.ui.perspectiveExtensions">
```

```
      <perspectiveExtension
            targetID="org.eclipse.jdt.ui.PackageExplorer">
         <view
               ratio="0.5"
               relative="org.eclipse.ui.views.ResourceNavigator"
               relationship="right"
               id="pluginvista.views.Vista">
         </view>
      </perspectiveExtension>
   </extension>
```

Esta extensión sirve para crear las llamadas a las funciones que activara cada botón que creamos. Las funciones de estos botones se detallaran más adelante

```
      <extension
         point="org.eclipse.ui.commands">
      <category
            name="Visualizar"
            id="Pluginboton.commands.category">
      </category>
      <command
            name="Visualizar"
            categoryId="Pluginboton.commands.category"
            id="Pluginboton.commands.Handler">
      </command>
   </extension>
   <extension
         point="org.eclipse.ui.handlers">
      <handler
            commandId="Pluginboton.commands.ComandoHandler"
            class="pluginboton.handlers.Handler">
      </handler>
   </extension>
     <extension
         point="org.eclipse.ui.handlers">
      <handler
            commandId="Pluginboton.commands.HandlerClean"
            class="pluginboton.handlers.HandlerClean">
      </handler>
   </extension>
```

Esta extensión añade al menú superior opciones con el título AEDViewer y un submenú con “Visualizar” y “Limpiar”.

```
   <extension
         point="org.eclipse.ui.menus">
      <menuContribution
            locationURI="menu:org.eclipse.ui.main.menu?after=additions">
```

```
         <menu
               label="AEDViewer"
               id="Pluginboton.commands.Visualizar">
            <command
                  commandId="Pluginboton.commands.ComandoHandler"
                  icon="icons/icon_component.gif"
                  id="Pluginboton.commands.comando"

                  label="Visualizar">
            </command>
             <command
                  commandId="Pluginboton.commands.HandlerClean"
                  icon="icons/delete.gif"
                  id="Pluginboton.commands.comandoClean"
                  label="Limpiar">
            </command>
         </menu>
      </menuContribution>
      <menuContribution

locationURI="toolbar:org.eclipse.ui.main.toolbar?after=additions">
         <toolbar
               id="Pluginboton.toolbars.sampleToolbar">
            <command
                  commandId="Pluginboton.commands.ComandoHandler"
                     icon="icons/icon_component.gif"
                  tooltip="Visualizar"
                  id="Pluginboton.toolbars.comandoToolbar">
            </command>
         </toolbar>
      </menuContribution>
       <menuContribution

locationURI="toolbar:org.eclipse.ui.main.toolbar?after=additions">
         <toolbar
               id="Pluginboton.toolbars.sampleToolbar">
            <command
                  commandId="Pluginboton.commands.HandlerClean"
                     icon="icons/delete.gif"
                  tooltip="Limpiar"
                  id="Pluginboton.commands.comandoCleanToolbar">
            </command>
         </toolbar>
      </menuContribution>
   </extension>
```

5 ANÁLISIS FUNCIÓNAL

Una vez especificados los requisitos daremos a conocer las funcionalidades en detalle y los flujos para ellas. Primero se conocerán los casos de uso, en el que se puede ver la interacción entre el sistema y el actor que en nuestro caso al ser un plugin para una

herramienta de ordenador personal tendremos un solo actor que realizara las acciones siendo este el usuario de eclipse.

5.1 CASOS DE USO

A continuación, se detallan los casos de uso del plugin AED-Viewer.

5.1.1 Caso de uso 1: Pintar figura

El usuario quiere mostrar una estructura en la vista para ello debe tener activado el depurador. Si no está activado al intentar pintar le mostrará un mensaje de que no tiene activado el depurador y no podrá realizar ninguna acción hasta que lo tenga activado.

Cuando este activado el depurador al pulsar en el botón de “Visualizar” le aparecerá un listado con las variables de Datos Abstractos que se han leído hasta el punto de ruptura al que haya llegado el depurador. El usuario al seleccionar una variable le aparecerá en la vista “Visualizador” con el tipo de dibujado para esa estructura.

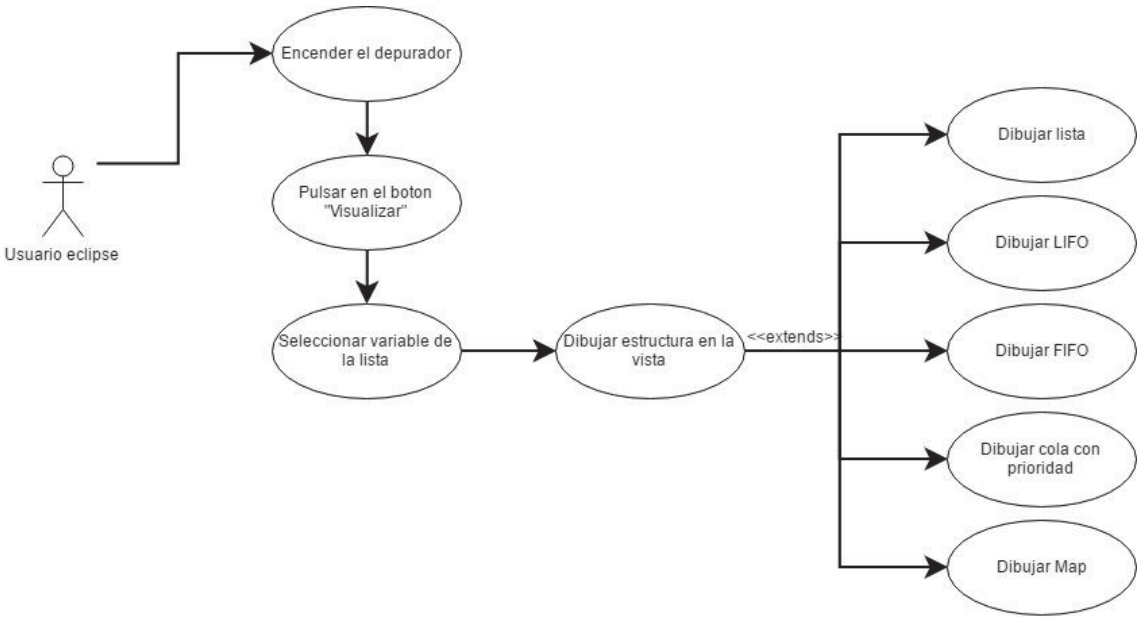

Ilustración 18 Primer caso de uso

5.1.2 Caso de uso 2: Mostrar el contenido de un elemento de la estructura

Para mostrar el contenido de una variable primero el usuario debe haber tenido un objeto de tipo abstracto de datos que haya podido leer el plugin para poder visualizarla en la lista que mostrará al usuario y pueda seleccionarla. Una vez seleccionada lo mostrara en la vista ·Visualizar”.

Ilustración 19 Segundo caso de uso

5.1.3 Caso de uso 3: Limpiar cualquier contenido que este en la vista

Si el usuario desea eliminar el contenido de la vista "Visualizar" puede hacerlo desde el botón Limpiar o desde el menú la pestaña AEDViewer submenú Limpiar. No es necesario tener el depurador abierto para esta acción.

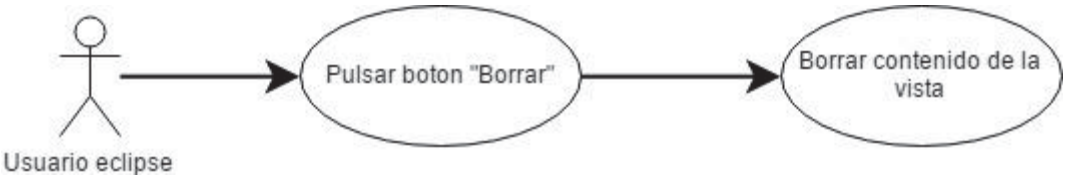

Ilustración 20 Tercer caso de uso

6 DISEÑO TÉCNICO

Una vez explicado la interacción del sistema con el usuario, se verá que clases actúan para los distintos casos de uso y la función que desempeña.

Antes de hablar de las clases hay que mencionar un fichero por el cual se configura todo el plugin este fichero es plugin.xml mencionado anteriormente.

Además de la construcción del fichero plugin.xml hay que destacar otro modulo que hay que construir para el correcto funcionamiento del plugin. Se trata del fichero que se compilara para importar las librerías necesarias para crear el plugin.

MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: PluginVPN
Bundle-SymbolicName: PluginVPN;singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: plugin.Activador.Activator
Require-Bundle: org.eclipse.core.runtime,
 org.eclipse.ui,
 org.eclipse.core.resources;bundle-version="3.11.1",
 org.eclipse.swt,
 org.eclipse.osgi,
 org.eclipse.debug.ui,
 org.eclipse.jdt.debug;bundle-version="3.10.1"
Bundle-RequiredExecutionEnvironment: JavaSE-1.8
Import-Package: org.eclipse.core.resources
Bundle-ActivationPolicy: lazy
Bundle-ClassPath: .,
 lib/net-datastructures-5-0.jar,
```

```
lib/aedlibraries.jar
```

Como podemos observar usamos las librerías "`lib/net-datastructures-5-0.jar`" y "`lib/aedlibraries.jar`" que son las librerías que usamos para manejar las estructuras de datos creadas por el departamento.

6.1 Diagrama de Clases

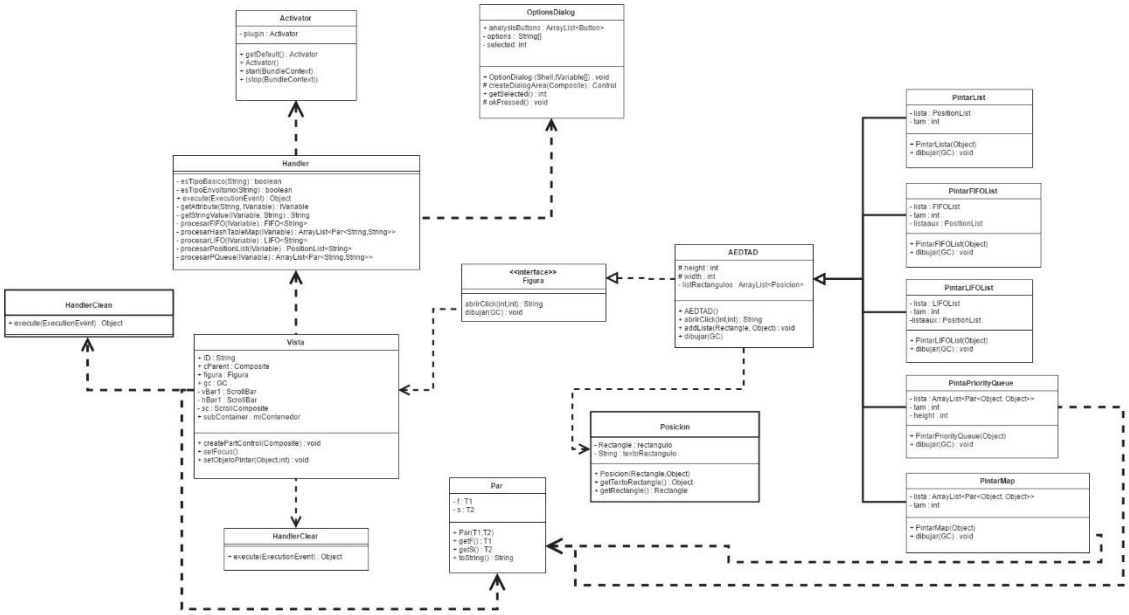

Ilustración 21 Diagrama de clases

En el siguiente diagrama podemos ver las relaciones entre clases: A continuación, comentaremos las clases:

- **Handler**: Es la clase que controla y gestiona las llamadas a las demás clases.
- **HandlerClean**: Es la clase que gestiona el limpiado de la vista.
- **Activator**: es la que se encarga de inicializar el plugin ya que tiene los métodos para ello.
- **OptionsDialog**: Se encarga de mostrar y recoger la selección del usuario de la variable leída por Handler. Luego se comunica con la vista pasándole la variable seleccionada.
- **Vista**: se encarga de gestionar el dibujado y hacer lo que corresponda. Vista se encarga de dibujar la estructura y de escuchar el ratón para cuando seleccione un punto en la vista. Figura es la interfaz de la que hereda las estructuras, lista, FIFO, LIFO, priority queue, Map … y estas tienen su propio método para dibujar la estructura de la forma en la que se haya integrado.

- **AEDTAD**: guarda es una lista las posiciones de los objetos de la vista Visualizador. Una posición contiene un objeto rectángulo y un texto con el valor del mismo.
- **HandlerClean**: Se encarga de borra el contenido presentado en la vista
- **Par**: Clase de apoyo que guarda dos atributos para modelar el tipo de objeto Entry<,> que utilizaremos para el dibujado de mapas y colas con prioridad

6.2 Diagrama de Secuencia

Se detallará el ciclo que se sigue por las distintas clases para realizar los casos de uso citados anteriormente.

El primer diagrama de secuencia muestra las clases que interfiere para dibujar los distintos tipos de datos en la vista. El usuario al iniciar la acción de visualizar a través del botón o el submenú. Handler se encarga de llamar al depurador y comprobar si está activado, si es así lee las variables de la vista "Variables" y procede a categorizar cuales son los tipos aptos para la visualización. Una vez que tenemos los datos aptos OptionDialog se encarga de mostrarle al usuario las variables y permitir elegir una variable. La variable elegida se devuelve a la clase Handler y procesara los datos de esa variable para pasárselo a la Vista. Una vez que la vista tiene esa variable se encarga de dibujarla en la vista Visualizador.

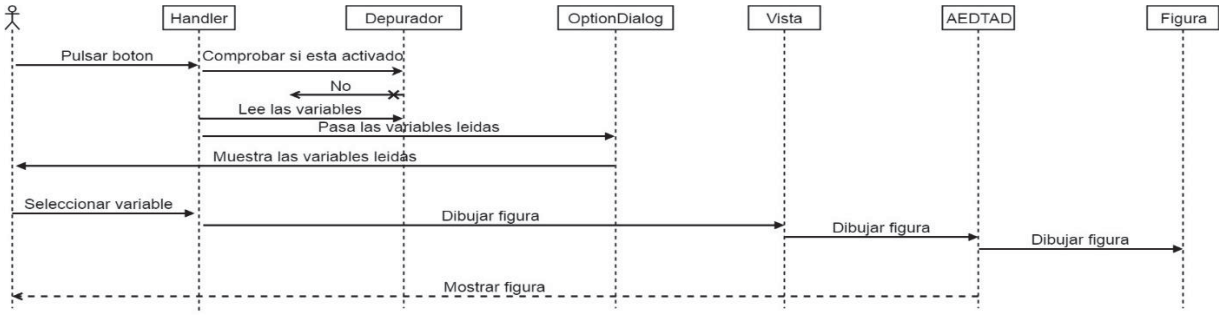

Ilustración 22 Diagrama de secuencia del Caso de Uso 1

Se ha limitado el diagrama de secuencia hasta Figura, pero hay que tomar en cuenta que Figura llama al método correspondiente para pintar cada tipo abstracto de datos.

El segundo diagrama de secuencia trata sobre la visualización del contenido de los datos de uno de los objetos pintados en la vista Visualizador. El usuario al dar clic izquierdo con el ratón en una zona de la vista, la vista activara un evento por el cual comprueba si la zona pulsada pertenece al área del objeto que se ha pintado, si es así muestra el texto de ese objeto en una nueva ventana con toda la información del objeto pulsado.

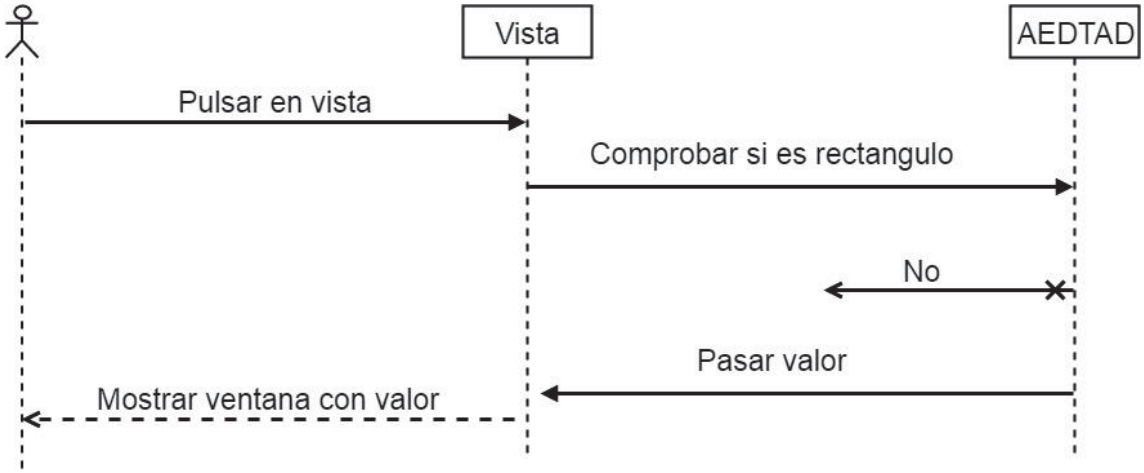

Ilustración 23 Diagrama de secuencia del Caso de Uso 2

El tercer diagrama de secuencia limpia la ventana "Visualizador" de cualquier dibujo que estuviese mostrando. El usuario al accionar Limpiar pulsando el icono en la barra toolbar o en el menú creado llamara un evento que borre el contenido de la vista "Visualizador".

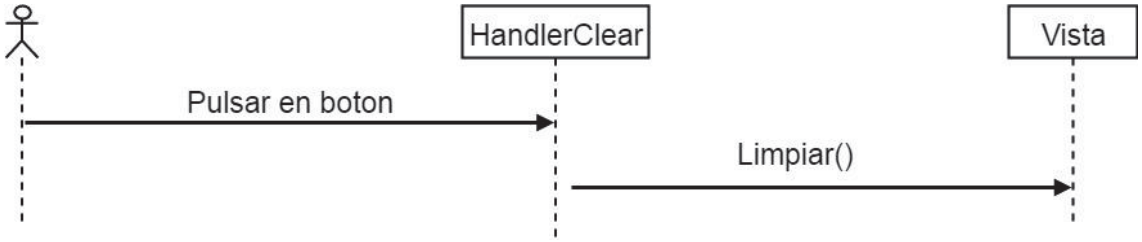

Ilustración 243 Diagrama de secuencia del caso de Uso 3

6.3 Detalle de las clases

A continuación, se muestra la estructura de paquetes del proyecto, con cada una de las clases comentadas con anterioridad.

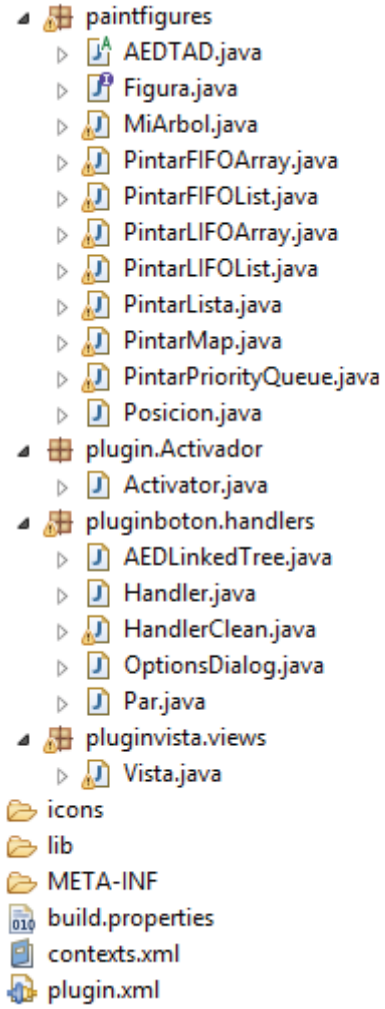

Ilustración 25 Listado de clases desde el proyecto

A continuación, se detalla el contenido de cada una de las clases, así como el comportamiento de cada uno de los métodos de las mismas.

6.3.1 Pluginvista.Vista

Clase para la creación de la vista y los eventos que se producirán desde la vista.

Herencia y características

Extiende de la clase ViewPart.

Atributos

Nombre	Descripción	Tipo	Visibilidad
subContainer	Contenedor para el dibujado	Composite	
gc	Panel para el dibujado	GC	
sc	Crea el contendor	ScrolledComposite	

	de las barras de scroll		
figura	Recoge la figura	Figura	
cParent	Variable que copia el composite padre	Composite	
ScrollBar	Scrollbar vertical	VBar1	
ScrollBar	Scrollbar horizontal	HBar1	

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
setFocus	Función que se ejecuta cunado se hace foco en la vista	Ninguno	public	Nada
createPartControl	Función que se ejecuta al crear la vista	Composite parent	Public	Nada
setObjectoPintar	Comprueba de que tipo es el dato que le ha pasado el handler y copia el objeto a Figura	Object miestructura int tipo	public	Nada

6.3.1.1 Método setFocus

Argumentos: Ninguno

Retorno: Nada

Descripción: Función de ViewPart obligatoria ponerla, ejecuta el código de dentro de la función cuando haces el foco en la vista

6.3.1.2 Método createPartControl

Argumentos: Composite parent → Recibe el contenedor padre de la vista

Retorno: Nada

Descripción: Al crear la vista ejecuta los métodos que tiene internos. Crea internamente un contenedor con GC este implementa un scroll y todo ello unido a otro contenedor, también crea los listener del mouse, para identificar cuando se está haciendo click en la ventana y un listener de dibujado para pintar cada vez que necesite cambiar de estructura

6.3.1.3 Método setObjetoPintar

Argumentos: Object miestructura → La estructura a mostrar

Int tipo → Recibe el tipo de la estructura

Retorno: Nada

Descripción: Comprueba el tipo mediante un switch, si identifica el tipo crea una nueva instancia de ese objeto en Figura

6.3.2 Paintfigures.AEDTAD

Herencia y características

Implementa Figura para contener los rectángulos presentados y gestionar el click del usuario.

Atributos

Nombre	Descripción	Tipo	Visibilidad
width	Ancho del rectángulo	int	
height	Altura del rectángulo	int	
listRectangulos	Vector de posiciones	ArrayList<Posicion>	

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
AEDTAD	Constructor	Ninguno	public	Nada
addLista	Añade un elemento a la lista	Rectangle Rect , Object text	Public	Nada
abrirClick	Recorre la lista comprobando si en esa posición hay guardado un objeto	Int x, int y	public	String

6.3.2.1 Método AEDTAD

Argumentos: Ninguno

Retorno: Nada

Descripción: Inicializa la arrayList de posiciones.

6.3.2.2 Método addLista

Argumentos: Rectangle Rect → Variable rectángulo que contiene su posición, así como su ancho y altura

Object text → Texto que va a guardar el rectángulo de esa posición

Retorno: Nada

Descripción: Añade un elemento a la lista de posiciones

6.3.2.3 Método addLista

Argumentos: int x → Posición del eje x en el que se ha pulsado

Int y → Posición del eje y en el que se ha pulsado

Retorno: Una cadena de texto con el valor del rectángulo

Descripción: Recorre la lista de posiciones y comprueba si donde se ha pulsado existe un rectángulo y coge el valor que tiene ese rectángulo

6.3.3 Paintfigures.Figura

Es un interfaz que dispone de los métodos para el dibujado de la figura, aunque luego cada figura debe tener su propia implementación.

Herencia y características

Dispone de los métodos para el dibujado de la figura

Atributos

No contiene ningún atributo

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
dibujar	Dibuja la estructura en la lista	GC gc	Public	Nada
abrirClick	Recorre la lista comprobando si en esa posición hay guardado un objeto	Int x, int y	public	String

Al ser un interfaz estos métodos están explicados en las clases que lo implementan.

6.3.4 Paintfigures.PintarFIFOList

Clase para el dibujado de la estructura de FIFOList.

Herencia y características

Extiende AEDTAD

Atributos

Nombre	Descripción	Tipo	Visibilidad
lista	Es una cola FIFO	FIFO	
tam	Tamaño de la estructura	int	
listaaux	Lista para tranformar la FIFO en una positionList para poder moverse	PositionList	

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
PintarFIFOList	Constructor	Object miestructura	public	Nada
dibujar	Pinta la estructura	GC gc	Public	Nada

6.3.4.1 Método PintarFIFOList

Argumentos: Object miestructura → Estructura que se quiere dibujar

Retorno: Nada

Descripción: Inicializa la PositionList transformando la FIFO y además añade a la lista de posiciones cada elemento que vaya recorriendo.

6.3.4.2 Método dibujar

Argumentos: GC gc → Panel de dibujado

Retorno: Nada

Descripción: Recorre la lista que transformamos en el constructor y por cada elemento comprueba su longitud hace las trasformaciones si las necesita y pinta en la vista, por cada elemento hace las mismas comprobaciones hasta recorres toda la lista

6.3.5 Paintfigures.PintarLIFOList

Clase para el dibujado de la estructura de LIFOList

Herencia y características

Extiende AEDTAD

Atributos

Nombre	Descripción	Tipo	Visibilidad
lista	Es una cola LIFO	LIFOList	
tam	Tamaño de la estructura	int	
listaaux	Lista para transformar la FIFO en una positionList para poder moverse	PositionList	

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
PintarLIFOList	Constructor	Object miestructura	public	Nada
dibujar	Pinta la estructura	GC gc	Public	Nada

6.3.5.1 Método PintaLIFOList

Argumentos: Object miestructura → Estructura que se quiere dibujar

Retorno: Nada

Descripción: Inicializa la PositionList transformando la LIFO y además añade a la lista de posiciones cada elemento que vaya recorriendo.

6.3.5.2 Método dibujar

Argumentos: GC gc → Panel de dibujado

Retorno: Nada

Descripción: Recorre la lista que transformamos en el constructor y por cada elemento comprueba su longitud hace las trasformaciones si las necesita y pinta en la vista, por cada elemento hace las mismas comprobaciones hasta recorres toda la lista

6.3.6 Paintfigures.PintarLista

Clase para el dibujado de la estructura de lista

Herencia y características

Extiende AEDTAD

Atributos

Nombre	Descripción	Tipo	Visibilidad
lista	Es una cola FIFO	PositionList	
tam	Tamaño de la estructura	int	

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
PintarLista	Constructor	Object miestructura	public	Nada
dibujar	Pinta la estructura	GC gc	Public	Nada

6.3.6.1 Método PintarLista

Argumentos: Object miestructura → Estructura que se quiere dibujar

Retorno: Nada

Descripción: Inicializa la PositionList y va añadiendo a la lista de posiciones cada elemento que va recorriendo.

6.3.6.2 Método dibujar

Argumentos: GC gc → Panel de dibujado

Retorno: Nada

Descripción: Recorre la lista y por cada elemento comprueba su longitud, hace las trasformaciones si las necesita y pinta en la vista hasta terminar de recorrerla.

6.3.7 Paintfigures.PintarMap

Clase para el dibujado de la estructura de HashMap

Herencia y características

Extiende AEDTAD

Atributos

Nombre	Descripción	Tipo	Visibilidad
lista	Es un arrayList que contiene objetos de tipo Par que intenta simular una objeto con clave y valor	ArrayList<Par<Object,Object>>	
tam	Tamaño de la estructura	int	

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
PintarMap	Constructor	Object miestructura	public	Nada
dibujar	Pinta la estructura	GC gc	Public	Nada

6.3.7.1 Método PintarMap

Argumentos: Object miestructura → Estructura que se quiere dibujar

Retorno: Nada

Descripción: Inicializa un arrayList con la estructura de tipo Par, separa los dos objetos y va añadiendo a la lista de posiciones cada par de elementos que va recorriendo.

6.3.7.2 Método dibujar

Argumentos: GC gc → Panel de dibujado

Retorno: Nada

Descripción: Recorre la lista y por cada elemento comprueba su longitud, hace las trasformaciones si las necesita y pinta en la vista hasta terminar de recorrerla.

6.3.8 Paintfigures.Posicion

Clase para guardar el rectángulo y el texto correspondiente para ese rectángulo

Herencia y características

No tiene ninguna clase implementada o extendida

Atributos

Nombre	Descripción	Tipo	Visibilidad
rectángulo	Rectángulo que se ha pintado	Rectangle	
textorectangulo	Cadena de caracteres que se saca del elemento que se está consultando	Object	

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
Posicion	Constructor	Rectangle rectángulo, Object textorectangulo	public	Nada
getRectangulo	Devuelve el rectángulo	Nada	Public	Rectangle
getTextoRectangulo	Devuelve el texto del rectángulo		Public	String

6.3.8.1 Método Posicion

Argumentos: Rectangle rectángulo → Rectángulo que se quiera instanciar

Object textoRectangulo → Texto asociado al rectángulo

Retorno: Nada

Descripción: Construye el rectángulo y el texto que pertenece a ese rectángulo

6.3.8.2 Método getRectangulo

Argumentos: Ninguno

Retorno: Nada

Descripción: Devuelve el rectángulo

6.3.8.3 Método getTextoRectangulo

Argumentos: Ninguno

Retorno: Nada

Descripción: Devuelve el texto del rectángulo

6.3.9 Pluginboton.handler.HandlerClean

Escuchador del botón “Borrar”

Herencia y características

Extiende de AbstractHandler

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
execute	Función que se ejecuta al dar clic en el botón borrar	ExecutionEvent event	public	Object

6.3.9.1 Método execute

Argumentos: ExecutionEvent event → Evento interno que se obtiene al pulsar el botón

Retorno: Object → No se utiliza

Descripción: Activa la función de la vista para borrar el contenido

6.3.10 Pluginboton.handler.OptionDialog

Comprueba las variables que se han leído del depurados y muestra en una lista las variadles del tipo de estructuras que se han realizado para este trabajo.

Herencia y características

Extiende de Dialog

Atributos

Nombre	Descripción	Tipo	Visibilidad
selected	Guarda la variable que se ha seleccionado	IVariable	Private
options	Lista de variables que se han leído del depurador	ArrayList<IVariable>	private
analysisButtons	Lista de botones que se crea al leer las variables del depurador	ArrayList<Button>	private

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
checkType	Comprueba si es de un determinado tipo disponible	String type	public	boolean
OptionsDialog	Crea la lista de variables con las variables que se han leído del depurador	Shell parentShell, IVariable[] options	public	
CréateDialogArea	Muestra en un panel la lista de variables	Composite parent	protected	Control
okPressed	Comprueba que se ha seleccionado una variable de la lista	Ninguno	protected	
getSelected	Devuelve el valor selecionado	Ninguno	public	IVariable

6.3.10.1 Método checkType

Argumentos: String type → Cadena de texto con el tipo de la variable

Retorno: boolean → Resultado de la comprobación de que la variable es uno de los tipos disponibles

Descripción: Comprueba que cada variable que es leída es de uno de los tipos de estructuras de AED

6.3.10.2 Método OptionsDialog

Argumentos: Shell parentShell, → Panel para visualizar

IVariable[] options → Las variables leídas del depurador

Retorno: Nada

Descripción: Crea los botones de la lista de variables leídas del depurador además de llamar a la función que comprueba si es un tipo de las estructuras de AED.

6.3.10.3 Método CréateDialogArea

Argumentos: Composite parent → Recibe el contenedor padre del eclipse

Retorno: Control → No se utiliza

Descripción: Crear la ventana con la lista de las posibles variables

6.3.10.4 Método okPressed

Argumentos: Ninguno

Retorno: Ninguno

Descripción: Comprueba si se ha pulsado en una de la lista de las variables, guarda la variable pulsada y cierra la ventana

6.3.10.5 Método getSelected

Argumentos: Ninguno

Retorno: IVariable → Valor pulsado de la lista

Descripción: Devuelve el valor pulsado en la lista

6.3.11 Pluginboton.handler.Handler

Escuchador del botón "Visualizar"

Herencia y características

Extiende de AbstractHandler

Métodos

Nombre	Descripción	Argumentos	Visibilidad	Retorno
execute	Función que se ejecuta al dar clic en el botón visualizar	ExecutionEvent event	public	Object
EsTipoBasico			Private	Boolean
EsTipoEnvoltorio			Private	Boolean
GetAttribute		String name, IVariable variable	Private	IVariable
GetStringValue		IVariable var, String espacios	Private	String
ProcesarFIFO		IVariable fifo	Private	FIFO<String>
ProcesarHashTable		IVariable map	Private	ArrayList<Par<String,String>>
ProcesarLIFO		IVariable fifo	Private	LIFO<String>
ProcesarPositionlist		IVariable list	Private	PositionList<String>
procesarPQueue		IVariable pqueue	private	ArrayList<Par<String,String>>

6.3.11.1 Método execute

Argumentos: ExecutionEvent event → Evento interno que se obtiene al pulsar el botón

Retorno: Object → No se utiliza

Descripción: Activa la función de la vista para dibujar la estructura que se haya seleccionado anteriormente

6.3.11.2 Método EsTipoBasico

Argumentos: Ninguno

Retorno: Boolean

Descripción: Comprueba si la variable es de un tipo genérico y no un objeto personalizado

6.3.11.3 Método EsTipoEnvoltorio

Argumentos: Ninguno

Retorno: Boolean

Descripción: Comprueba si la variable es de un tipo genérico y no un objeto personalizado

6.3.11.4 Método GetAttribute

Argumentos: String name, →

IVariable variable →

Retorno: IVariable

Descripción:

6.3.11.5 Método GetStringValue

Argumentos: IVariable var, →Variable para transformar

String espacios →Variable con un numero de espacios

Retorno: String →

Descripción: Convierte la variable en una serie de Strings para proyectar una estructura con sus distintas variables

6.3.11.6 Método ProcesarFIFO

Argumentos: IVariable fifo →Variable para transformar

Retorno: FIFO<String>

Descripción: Obtiene la composición interna de la estructura y lo convierte en una estructura del tipo FIFO para poder procesarla en la vista

6.3.11.7 Método ProcesarHashTable

Argumentos: IVariable map →Variable para transformar

Retorno: ArrayList<Par<String,String>>

Descripción: Obtiene la composición interna de la estructura y lo convierte en una estructura del tipo Map para poder procesarla en la vista

6.3.11.8 Método ProcesarLIFO

Argumentos: IVariable fifo →Variable para transformar

Retorno: LIFO<String>

Descripción: Obtiene la composición interna de la estructura y lo convierte en una estructura del tipo LIFO para poder procesarla en la vista

6.3.11.9 Método ProcesarPositionlist

Argumentos: IVariable list →Variable para transformar

Retorno: PositionList<String>

Descripción: Obtiene la composición interna de la estructura y lo convierte en una estructura del tipo PositionList para poder procesarla en la vista

6.3.11.10 Método procesarPQueue

Argumentos: IVariable pqueue →Variable para transformar

Retorno: ArrayList<Par<String,String>>

Descripción: Obtiene la composición interna de la estructura y lo convierte en una estructura del tipo Priority Queue para poder procesarla en la vista

7 DESARROLLO

Al empezar el trabajo de fin de grado lo primero que hubo que abordar fue el plan de proyecto y como organizaríamos las semanas. Esto dio una visión de que rumbo iba a tomar el proyecto a medida de como pasaban los días, así como las previsiones de los objetivos creados y los cambios que se realizarían más adelante. Lo podemos representar mediante el diagrama de Gantt del informe intermedio ya que tuvimos que hacer modificaciones por los problemas encontrados y la envergadura del proyecto.

Id	Nombre de tarea	Trabajo	Comienzo	Fin	Predecesoras
1	Redacción de la memoria del proyecto	31 horas	mar 28/02/17	mar 06/06/17	
2	Preparar presentación final	10 horas	mié 07/06/17	jue 22/06/17	
3	Preparar y redactar la memoria de seguimiento	6 horas	mié 05/04/17	vie 21/04/17	
4	Redactar el plan de proyecto	8 horas	lun 13/02/17	jue 16/02/17	
5	Estudio y evaluación del desarrollo de un plugin para Eclipse	15 horas	lun 13/02/17	sáb 18/02/17	
6	Análisis de posibles librerías de dibujado y su integración con eclipse	10 horas	sáb 18/02/17	mié 22/02/17	5
7	Preparación y estudio de la herramienta de dibujado	20 horas	mié 22/02/17	mié 01/03/17	6
8	Diseñar, planificar y organizar la representación de las estructuras de datos	100 horas	mié 01/03/17	vie 07/04/17	7
9	Integración del plugin en Eclipse con la herramienta de dibujado	90 horas	vie 07/04/17	sáb 13/05/17	8
10	Estudio de las estructuras de datos impartidas en la asignatura y dadas en su propia librería	5 horas	sáb 13/05/17	dom 14/05/17	9
11	Pruebas unitarias y de integración con el sistema	40 horas	dom 14/05/17	dom 28/05/17	10

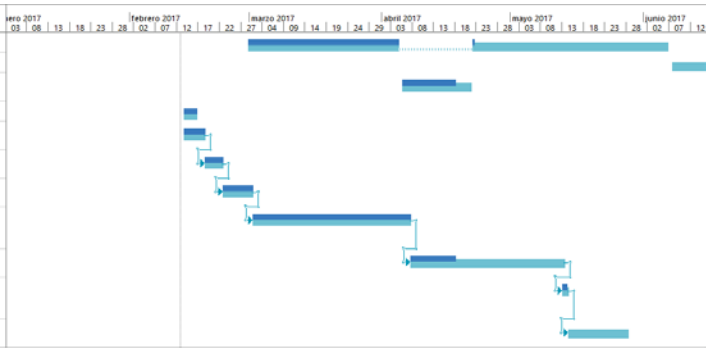

Ilustración 26 Diagrama de Gantt

Se realizaron reuniones semanales para ver los avances y los objetivos de la siguiente semana, así como las dudas que se tuvieran para el desarrollo. También se realizó un repositorio compartido para tener un proyecto compartido a través de Subversion en el que se podían ver los cambios realizados por los usuarios. A medida que se avanzaba la hora de la entrega de la presente memoria nos fuimos reuniendo más de lo planificado para concretos puntos tanto de la memoria como del código.

El primer paso para poder realizar el proyecto fue el estudio de RCP del acrónimo de Rich Client Platform. RCP es la arquitectura de Eclipse y que permite el desarrollo de nuevas funcionalidades. Para empezar, se creó un plugin básico en el que se añadió distintos elementos como botones vistas opciones de menú y mostraba mensajes en ventana. Esto sirvió para manejar las funcionalidades de cada elemento y ver los eventos que se lanzaban a medida que interactuabas.

El segundo paso fue el estudio de la herramienta gráfica. En principio se usó Zest una extensión de GEF que permitía un dibujado fácil y sencillo, pero causaba problemas entre la importación de la librería en la parte del plugin como en las propias maquinas ya que no identifica bien la librería, esto hizo modificar el código para los componentes de la vista y dibujar de otro modo. Por tanto, se usó GC, ya que, aunque el dibujado es manual se integraba muy bien en la vista y no hacía falta muchas importaciones de librerías externas, por tanto, nos permitía un dibujado manual fácil y sencillo. Para comprobar si la librería era buena para poder seguir con el plugin se hicieron pequeñas pruebas de dibujar elementos e interactuar con botones para crear y borrar elementos.

Una vez comprobado que se integraba bien se pasó a dibujar las estructuras desde la propia vista.

Para dibujar las estructuras primero nos reunimos Guillermo y yo para estudiar como pintar las estructuras con lo que nos ofrecía GC y que las estructuras fueran suficientemente claras para evitar confusiones a los usuarios. También se tuvo que estudiar las distintas funciones que ofrecían cada estructura tanto para moverme por ella como para crearlas.

Uno de los problemas fue el árbol ya que el dibujado era difícil y costoso por tanto se propuso implementarlo en la pestaña de outline del propio Eclipse. Se estudió la posibilidad, pero al igual que el dibujado con GC era muy difícil implementarlo ya la vista Outline solo trabaja con la parte del editor y no con objetos. También fue difícil buscar implementaciones sobre el dibujado de árboles ya que nadie lo había hecho antes o al menos mostrado en Internet. Para la creación del árbol se usó parte de una implementación de un usuario.

Al crear todo el código en la propia vista estábamos usando un lenguaje estructurado en el que todo el código se concentraba en una misma clase. Por ello se realizó una reunión para organizar las partes del proyecto, esto hizo que hubiera que retocar de nuevo varias funciones ya que dejaron de funcionar al implementarlo en una clase distinta de la que ya había.

Al estudiar la interacción entre la vista del debug y nuestro plugin se descubrió que no se podían comunicar ya que actuaban en máquinas virtuales distintas. Para solucionarlo se obtenía las referencias de los objetos y se recorría internamente el objeto para conseguir las construir una versión de ese mismo objeto en el orden que recorrías internamente el objeto de la vista de variables.

Una vez que ya conseguíamos las variables se programaros los botones que de tal forma leyera las variables que fuera leyendo y poder seleccionar una para que pintara. Previamente se tuvo que arreglar el repintado ya que se las figuras se solapaban si se cambiaba de estructura, esto se hizo con un reajuste de código en el que se eliminaban muchos de los escuchadores haciendo que además se redujera el código bastante.

Mientras se desarrollaba la extracción de las estructuras se optimizo el código eliminando código innecesario, así como optimizando el código principal.

Para continuar se hicieron mejoras en la visualización de los textos ya que si la estructura contenía otros objetos no se diferenciaba de cuando eran variables, así como si tenía muchos datos o eran demasiados extensos por tanto se acortaron los textos y se indicó si el valor de la variable era más extenso se añadió en modo indicativo puntos suspensivos para que el usuario pudiera saber que el valor era mayor. También los puntos suspensivos si aparecían más abajo significa que tiene más datos, pero por el tamaño del objeto no se reproducirían en la imagen. Estas dos cuestiones ultimas cuestiones con un simple clics al objeto mostraría tanto el valor de la variable, así como

todas las variables que tiene dentro. Para la muestra de variables dentro de otras variables se tabula dos espacios hacia la derecha para permitir hacer una representación de un árbol.

Por último, se creó la función borrar para facilitar al usuario la limpieza de la vista. Esto fue sencillo después de haber creado la función de "Visualizar".

8 MANUAL DE USUARIO

Esta sección servirá para guiar al usuario al aprendizaje del uso del plugin de Eclipse.

Para empezar, debemos comprobar si tenemos la vista Visualizador en nuestra perspectiva del depurador.

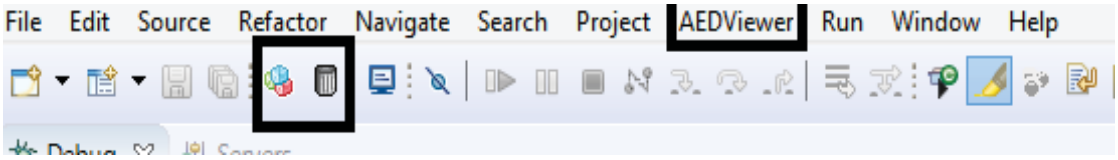

Ilustración 27 Menú y botones plugin

Como podremos observar en la imagen se diferencian dos secciones si el plugin está activado tendremos en el menú superior AEDViewer y en la barra de herramientas dos iconos (Limpiar) (Visualizar).

Una vez comprobado que los tenemos tendremos que enseñar la vista en nuestra barra de vistas para ello iremos a menú → window → show view → other

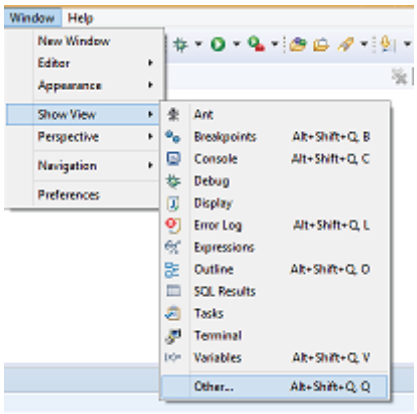

Ilustración 28 Mostrar vista 1

Nos aparecerá una ventana en el que tendremos que buscar la sección visualizador de estructuras y hacer doble clic en "Visualizador", también puedes buscar en el buscador superior visualizador.

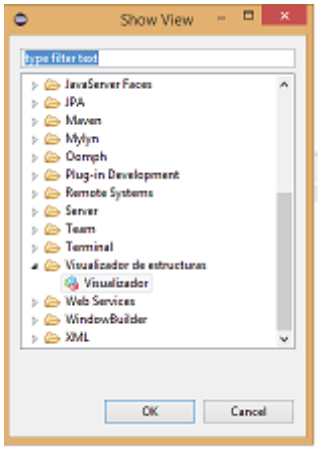

Ilustración 29 Mostrar vista 2

Podremos observar que la vista ha aparecido en un menú de vistas probablemente en la parte inferior de nuestro eclipse

Ilustración 30 Vista en la ventana

Si intentáramos dibujar sin haber encendido el depurador nos mostrara un mensaje avisándonos de que estamos depurando el programa

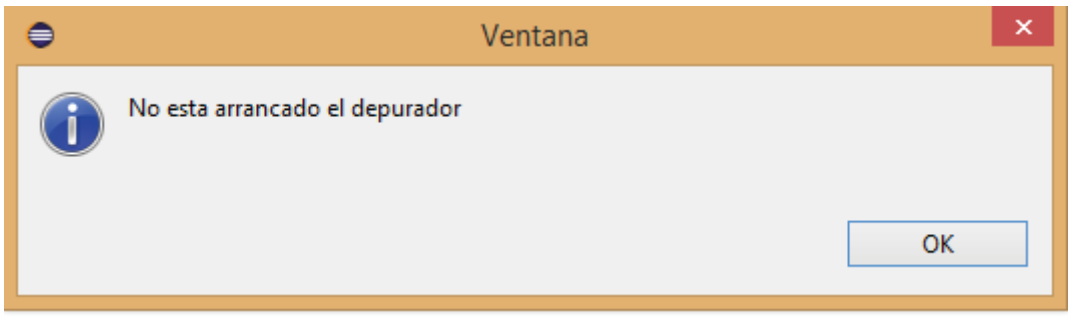

Ilustración 31 Mensaje de no activado depurador

Al encender el depurador también es necesario poner un punto de ruptura para que al parar el programa en el punto correspondiente tenga las variables que ha ido leyendo hasta ese punto de ruptura. Una vez que el programa ha llegado al punto de ruptura que deseamos podemos ejecutar el dibujar tanto por el submenú como por el icono

Ilustración 32 Icono visualizar

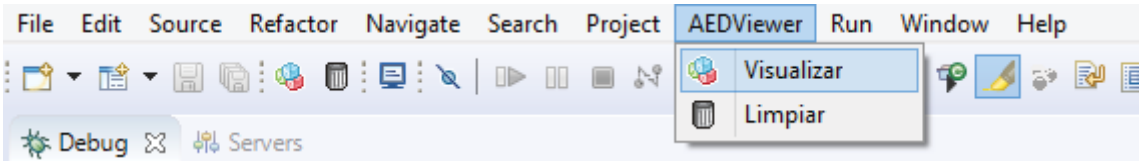

Ilustración 33 Menú visualizar

Una vez accionado si el depurador hubiese leído alguna de las estructuras de tipo abstracto implementadas aparecería una ventana con las variables que se han leído de ese tipo.

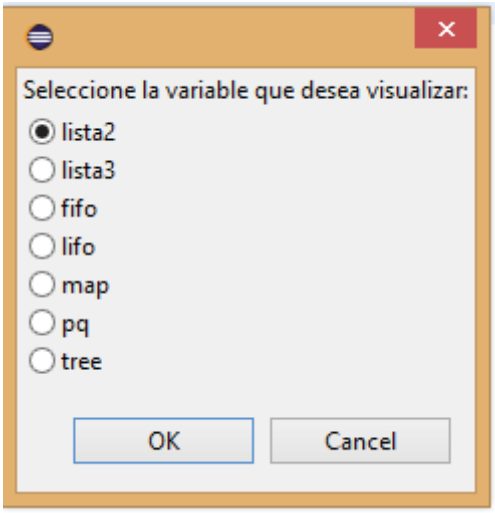

Ilustración 34 Lista variables

Al seleccionar una de las variables seleccionamos OK y nos dibujara en la vista la estructura correspondiente. En nuestro caso hemos elegido una vista y detallaremos la vista a continuación.

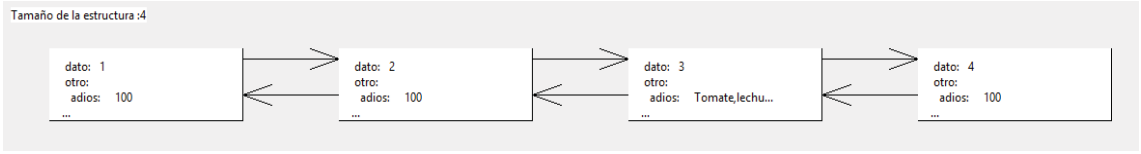

Ilustración 35 Dibujado de lista

Se puede observar en la esquina superior izquierda de la vista sale el tamaño de la estructura, esto se repite para todos los tipos de estructuras. Podemos observar que es

una lista ya que se dibuja de forma horizontal y se une a través de flechas en ambos sentidos ya que es una lista doblemente enlazada. Otra característica que podemos observar son los puntos suspensivos que aparecen en todos los rectángulos como 4º línea y los que aparecen en el rectángulo 3 en la línea tercera. Los primeros puntos son para reflejar que ese objeto tiene más variables dentro que no se están mostrando, los otros puntos son para destacar que esa variable tiene un valor largo para dibujarlo.

Si se selecciona el rectángulo 3 nos saldrá una ventana con la información del rectángulo completo con sus variables y valores independientemente de su longitud y el número de variables que tenga

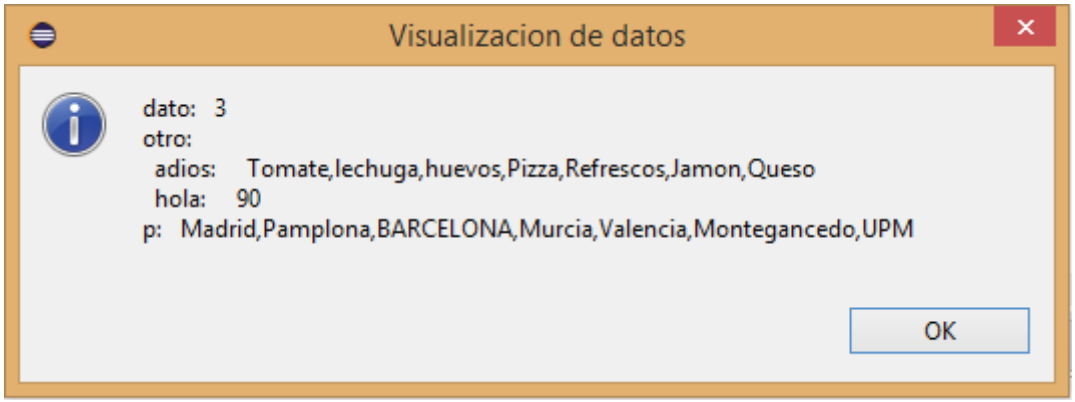

Ilustración 36 Mostrar datos de objeto

Si deseamos cerrar la ventana con seleccionar OK es suficiente.

Si se desea ver otra variable basta con volver a pulsar el botón visualizar y seleccionar la nueva variable o la misma si ha tenido algún cambio y se mostrara por la ventana.

Si queremos limpiar nuestra vista de cualquier dibujo que tenga solo hay que accionar "Limpiar" mediante el menú AEDViewer o el icono papelera

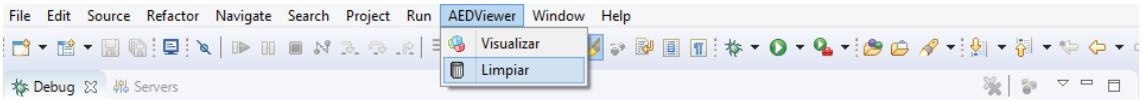

Ilustración 37 Menu limpiar

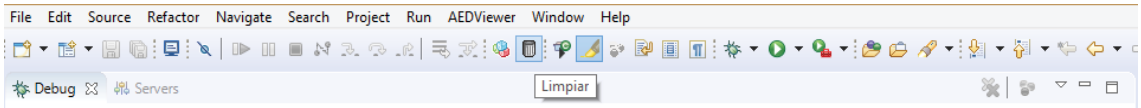

Ilustración 38 Icono limpiar

9 CONCLUSIONES

La realización de este proyecto me ha permitido aumentar mi experiencia tanto en la programación como en la parte de ingeniera del software en el desarrollo de documento funcional y técnico, asá como mejorar mi redacción. También me ha enseñado a realizar un proyecto desde cero por el cual conlleva muchas responsabilidades y sea necesaria la regularidad del desarrollo del proyecto y para así tener una flexibilidad en cuanto a cambios inesperados que se puedan producir. Por ello ha servido como una experiencia en que las estimaciones de cada punto en el proyecto pueden no ser las definidas, no por falta de planificación si no por los cambios que se pueden producir al no poder ir por un camino concreto por el que querías ir desarrollando parte de tu código y tener que buscar nuevas soluciones.

Hay que destacar que he mejorado mi nivel de programación y he conocido la herramienta Eclipse más a fondo por el cual me ha dado una visión de la complejidad interna de la herramienta.

El plugin puede ser una base para nuevas implementaciones y mejoras, algunas de ellas se podrían considerar en un futuro:

El plugin al depurar el programa y tener el punto de ruptura en un bucle en el que se modifica la estructura que se visualiza tiene que volver a visualizar el contenido pulsando en el botón de Visualizar o en el menú indistintamente se podría incorporar la automatización de la visualización de la misma estructura cuando esta se modifique sin pulsar en el botón.

Incorporar un botón en la vista que pase hasta el siguiente punto de ruptura para observar la modificación en la estructura. También se puede crear la estructura antes de la modificación tenían en la vista las dos estructuras una sin el cambio y otra con el cambio que se ha producido.

Extender el dibujado a las estructuras de tipos Abstractos de las librerías de Java y no solo las implementadas en la librería de la asignatura.

Incorporar el visionado del nodo al que estas seleccionando en el depurador para localizarlo en la estructura que estas pintando.

10 BIBLIOGRAFÍA

[1] http://help.eclipse.org/neon/index.jsp

[2] https://docs.oracle.com/javase/7/docs/api/overview-summary.html

[3] http://math.hws.edu/eck/cs124/f11/lab14/Rectangles.java

[4] https://vaadin.com/wiki/-/wiki/Main/Creating%20a%20simple%20component%20container

[5] http://www.java2s.com/

[6] http://www.dreamincode.net/

[7] http://www.javadocexamples.com/

[8] M T. Googrich and R Tamassia, Data Structures & Algorithms in Java, Fifth Edition, Wiley

[9] E Clayberg and D Rubel, Eclipse plug-ins, Third Edition, Eclipse

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Sun Jun 18 14:14:50 CEST 2017
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)