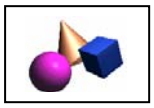
Sistemas Informáticos
Curso 2005-2006

Extensión de una herramienta para visualizar Estructuras de Datos y Algoritmos

Eduardo de la Iglesia Ramírez
Gonzalo Moreno Pereira
Cristina Rubert Sánchez

Dirigido por:
Prof. Clara María Segura Díaz
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid



RESUMEN EN CASTELLANO E INGLÉS

El objetivo del proyecto es la extensión de una herramienta pedagógica, destinada a los alumnos universitarios pertenecientes a las facultades de informática, que estén cursando las asignaturas de estructuras de datos y esquemas algorítmicos. Esta herramienta se ha usado como medio didáctico para facilitar el entendimiento de la asignatura de Estructuras de Datos y de la Información (2º Curso de Ingeniería Informática de la UCM) en este curso y se utilizará también en el futuro para ayudar a los alumnos en otras asignaturas, como Metodología y Tecnología de la Programación.

La extensión de la herramienta ha mejorado la estructura interna de la aplicación y ha añadido nuevas estructuras de datos como las tablas ordenadas, tablas hash abiertas, tablas hash cerradas y los primeros tipos abstractos de datos: un consultorio médico y un consultorio médico con prioridad. Estas estructuras son visualizadas de forma interactiva por los usuarios.

The aim of the project is to extend and to improve a pedagogical tool for students of a computer science degree who are taking courses in data structures and algorithmic schemes. This tool has been used this year as a didactic means in order to ease the understanding of a Data Structures course and will be used in the future to help students of other subjects, such as Programming Methodology.

The extension of the tool has improved the application’s internal structure and added new data structures like ordered tables, open hash tables, close hash tables and the first abstract data types: a medical doctor's office and a medical doctor's office with priority. These structures are visualized interactively by the users.

LISTA PALABRAS CLAVE

Acción: unidad necesaria para llevar a cabo la visualización y animación de una estructura de datos o algoritmo.

Algoritmo: conjunto de reglas que permiten obtener un resultado determinado a partir de ciertas reglas definidas.

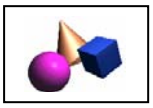
Animación: Conjunto de acciones destinadas a visualizar de forma animada el efecto producido tras la ejecución de las mismas sobre las estructuras de datos y algoritmos.

Esquema Algorítmico: plantilla proporcionada por una metodología de programación para resolver una colección de algoritmos de forma uniforme, que resuelven problemas específicos.

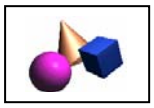
Estructuras de datos: colección de datos cuya organización se caracteriza por las funciones de acceso que se usan para almacenar y acceder a elementos individuales de datos.

Simulación: secuencia de acciones que muestran el comportamiento de forma animada sin necesidad de que el usuario introduzca los datos.

Visualización: efecto producido al mostrar en pantalla las estructuras de datos y algoritmos.



ÍNDICE



4.1. Introducción

Este documento es una especificación de Requisitos Software para la extensión de la herramienta "Visualización y Animación de Estructuras de Datos y Algoritmos". Esta especificación se ha estructurado basándose en las directrices dadas por el estándar "*IEEE recommended Practice for Software Requirements Specification ANSI/IEEE 830 1998*".

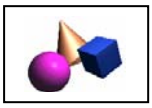
4.1.1. Alcance de la Aplicación

El objetivo del proyecto es la extensión de una herramienta pedagógica, destinada a los alumnos universitarios pertenecientes a las facultades de informática, que estén cursando asignaturas que tengan relación con las materias troncales de "Estructuras de Datos y de la Información" y "Metodología y Tecnología de la Programación". Esta herramienta ha sido usada en la asignatura de "Estructuras de Datos y de la Información" durante el curso 2005/2006 y será utilizada para la asignatura de "Metodología y Tecnología de la Programación" en el curso 2006/2007. Se pretende extender la herramienta para ampliar su funcionalidad y facilitar posibles extensiones futuras. El objetivo principal es usar dicha aplicación como método didáctico para facilitar el entendimiento y para promover el interés de los alumnos en estas materias.

La aplicación cuenta con una interfaz gráfica intuitiva sobre la que se ilustra el funcionamiento de diversas estructuras de datos y esquemas algorítmicos.

El usuario puede consultar o modificar cada una de las estructuras de datos disponibles, únicamente a través de las operaciones permitidas en las mismas. También puede ver la especificación algebraica de la estructura seleccionada, las diversas formas de implementación de las estructuras y el coste espacial y temporal de cada implementación.

Para cada uno de los esquemas algorítmicos se han desarrollado algoritmos concretos que resuelven problemas conocidos. Se pretende que los usuarios comprendan el funcionamiento de cada algoritmo concreto y por qué se usa un esquema algorítmico en particular para un determinado problema. Se permite al usuario visualizar el funcionamiento del algoritmo partiendo de datos introducidos por él mismo. Se puede visualizar la animación completa o bien paso a paso, permitiendo al usuario detenerla en todo momento para su completo entendimiento.



4.1.2. Propósito de la especificación

El objeto de la especificación es definir de manera clara y precisa todas las funcionalidades y restricciones del sistema que se desea construir. El documento va dirigido al equipo de elaboración del presente proyecto y a los usuarios finales del sistema. Este documento será el canal de comunicación entre las partes implicadas, tomando parte en su confección miembros de cada parte.

Esta especificación está sujeta a revisiones por el profesor y el equipo de elaboración, que se recogerán por medio de sucesivas versiones del documento, hasta alcanzar su aprobación. Una vez aprobado servirá de base para la construcción del nuevo sistema.

4.1.3. Visión General del Documento

La especificación de requisitos esta compuesta principalmente de dos secciones bien diferenciadas. Al tratarse de la extensión de una herramienta existente, primero se describe sin excesivo detalle la funcionalidad de la herramienta elaborada en el curso 2004/2005. La sección 4.2 pretende resumir las principales características de la aplicación, pudiéndose encontrar mayor detalle en el documento elaborado por Laura Gutiérrez García, Esther Rico Redondo y Carmen Torrano Giménez y cuyo nombre es “Visualización y Animación de Estructuras de Datos y Algoritmos”.

Seguidamente, en la sección 4.3, se ofrece una especificación de requisitos de la herramienta objetivo para el proyecto de la asignatura de Sistemas Informáticos con el fin de exponer las nuevas funcionalidades y las mejoras a realizar en el presente proyecto.

4.2. Descripción General de la herramienta 2004/2005

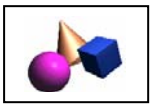
4.2.1. Descripción de la Funcionalidades del Sistema

Subsistemas de la Aplicación

El sistema esta compuesto de los siguientes subsistemas:

- Subsistema de Visualización y Animación de las Estructuras de Datos
- Subsistema de Visualización y Animación de los Algoritmos
- Subsistema de Documentación
- Subsistema de Simulación
- Subsistema de Ayuda

A continuación se comenta con más detalle qué características particulares posee cada uno de estos subsistemas.



Subsistema de Visualización y Animación de las Estructuras de Datos

En el subsistema de visualización y animación de las estructuras de datos se representa gráficamente y con animaciones cada una de las estructuras de datos que ofrece la herramienta, de modo que el usuario pueda observar el resultado de aplicar las distintas operaciones sobre cada estructura.

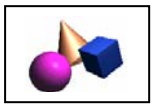
El subsistema cuenta con el siguiente conjunto de estructuras de datos:

- Pila
- Cola
- Árbol Binario de Búsqueda
- Árbol AVL
- Cola de Prioridad

A continuación se expondrá brevemente la funcionalidad de cada una de las vistas de las que dispone:

- **Vista usuario de la herramienta**: está destinada a usuarios que no conocen la implementación de cada estructura. Esta vista representa la estructura de forma independiente de su implementación interna, mostrando el resultado de cada operación permitida que el usuario realice sobre dicha estructura. Se da la opción de visualizar el estado actual de la estructura de datos, así como el estado anterior de la misma. Esta será la vista que aparecerá por defecto en la aplicación.
- **Vista usuario de desarrollo**: permite ver la representación gráfica de la implementación escogida de la estructura de datos. Se permite al usuario observar el funcionamiento interno de la estructura, así como comprobar la respuesta a determinadas operaciones. Dependiendo de la estructura de datos que estemos tratando puede haber varias vistas de desarrollo. Esto se debe a que se han realizado las implementaciones más comunes de cada estructura.

Todas las operaciones de las estructuras de datos pueden seleccionarse directamente en la interfaz o a través del menú. En el caso de que la función no pueda aplicarse por ser parcial, debido al estado de la estructura en ese momento, ésta aparecerá deshabilitada en el menú. Sin embargo, en la interfaz las funciones siempre estarán disponibles, y en caso de que la función no pueda aplicarse, aparecerá un mensaje que informe al usuario del suceso impidiendo llevar a cabo tal operación y no se producirá ningún cambio sobre la estructura de datos.



Subsistema de Visualización y Animación de los Esquemas Algorítmicos

En el subsistema de visualización y animación de Esquemas Algorítmicos se representa gráficamente con animaciones cada uno de los algoritmos que componen la herramienta, de modo que el usuario puede observar el proceso de ejecución del mismo. El usuario puede visualizar uno o varios ejemplos de cada uno de los esquemas algorítmicos.

El subsistema de gestión de algoritmos cuenta con el siguiente conjunto de esquemas algorítmicos:

- Voraz
- Programación Dinámica
- Divide y vencerás
- Ramificación y poda

En cada algoritmo se dispone de las siguientes funciones:

- **Introducir datos**: Permite al usuario introducir los datos de entrada con los que operará el algoritmo.
- **Iniciar**: Inicia la ejecución y animación del algoritmo escogido.
- **Pausar**: Interrumpe la ejecución y la animación del algoritmo hasta que vuelva a seleccionarse Ejecutar.
- **Ejecutar**: Reanuda la ejecución del algoritmo en el punto donde se quedó por última vez.
- **Parar**: Detiene y finaliza totalmente la ejecución y animación del ejemplo escogido.
- **Paso a paso**: Permite visualizar la animación de un paso de la ejecución del algoritmo.
- **Velocidad**: Esta función permitirá que se ejecute más deprisa o más despacio el algoritmo, incrementando o aminorando la velocidad.
- **Nuevos Datos**: Finaliza el algoritmo que se estuviera ejecutando en ese momento, limpia la pantalla y muestra la misma pantalla inicial que se visualiza al insertar los datos.

En cada una de las metodologías algorítmicas está implementado al menos un algoritmo que resuelve un problema concreto.

- Con el esquema algorítmico voraz, programación dinámica y ramificación y poda se resuelve el problema de la mochila (versión fraccionable en el voraz y entera en los otros dos) con el objetivo de que se pueda comparar el comportamiento de los diferentes esquemas algorítmicos ante el mismo problema.

- Usando el esquema algorítmico voraz también está implementado el algoritmo de Dijkstra para resolver el problema de caminos mínimos.
- Como ejemplos de la metodología de divide y vencerás están implementados los algoritmos de búsqueda binaria y ordenación rápida.

Para mayor información sobre las características de cada problema se recomienda consultar la memoria de la herramienta del curso 2004/2005 cuyo nombre es "Visualización y Animación de Estructuras de Datos y Algoritmos".

Subsistema de Documentación

El subsistema de documentación ofrece al usuario diversa información acerca de cada una de las estructuras de datos y esquemas algorítmicos.

Para cada ED se dispone de:

- La **especificación algebraica** de la estructura siguiendo como referencia el libro de Ricardo Peña [7].
- El **coste**: que contiene una tabla comparativa de los costes temporales y espaciales de las distintas operaciones de la estructura de datos, en función de la implementación utilizada en la misma.
- El **código fuente en Java** de la implementación de la estructura (el mismo utilizado en la aplicación).
- Además se dispone de una **ayuda adicional** que contiene información acerca de la estructura de datos concreta.

Para cada esquema algorítmico existe documentación con información acerca de:

- El **coste**: que contiene una tabla en la que podrá verse el coste del ejemplo implementado con el esquema algorítmico escogido y el coste de la implementación de ese mismo ejemplo con el resto de los esquemas algorítmicos aplicables a tal ejemplo. Esta tabla permite comparar los costes de los diferentes tipos de algoritmos, ya que se refieren al mismo ejemplo.
- El **código fuente en Java** del ejemplo implementado con el esquema algorítmico (el mismo utilizado en la aplicación).
- Además se dispone de una **ayuda adicional** que tiene información sobre las características del esquema algorítmico general.

Subsistema de Simulación

Este subsistema se encarga de todo lo referente a las simulaciones que se le proporcionan al usuario. Las simulaciones pretenden mostrar al usuario el comportamiento de las estructuras de datos y de los algoritmos de cada esquema algorítmico, sin necesidad de que el usuario introduzca a través de la interfaz las funcionalidades que desea realizar. Las simulaciones son una serie de ejemplos predeterminados en la aplicación, con ciertas funciones sobre las estructuras de datos y los datos de entrada de un determinado problema implementado con uno de los métodos algorítmicos. Consisten en ficheros de texto modificables que pueden ser usados desde la interfaz.

Subsistema de Ayuda

Este subsistema se hace cargo de todo lo relacionado con la ayuda proporcionada a los usuarios. Se permite al usuario la posibilidad de navegar en la ayuda hacia delante y hacia atrás, para que pueda retomar páginas de ayuda visitadas anteriormente.

Las funciones en que se descompone este subsistema son:

- **Índice:** desde el que se accede a toda la ayuda disponible en el sistema
- **Manual de usuario:** contiene las instrucciones requeridas para un correcto funcionamiento de la aplicación.

4.2.2. Perfiles de Usuario

Esta herramienta va dirigida fundamentalmente a alumnos de informática que cursen materias troncales similares a Estructuras de Datos y de la Información y Metodología y Tecnología de la Programación. Ésta aplicación también resultará útil a los profesores que imparten dichas asignaturas.

En este tipo de asignaturas es muy importante el trabajo continuado individual y la realización de múltiples ejercicios y ejemplos para poner en práctica los conocimientos adquiridos. Así se facilita la solución a los problemas tratados por estas asignaturas.

Esta herramienta puede resultar de gran utilidad y aportará grandes beneficios a los alumnos principiantes que se inicien en las asignaturas de EDI o MTP ya que su parte gráfica y visual ayuda a aclarar y comprender los conceptos.

No olvidemos que esta herramienta también puede servir de apoyo aquellos alumnos que estudian a distancia, por libre o con autoformación.

4.3. Requisitos Específicos de la herramienta 2005/2006

4.3.1. Estudio de requisitos

Los creadores de la herramienta del curso 2004/2005 propusieron en su memoria una serie de futuras ampliaciones funcionales y no funcionales para la herramienta:

- Implementar nuevas estructuras de datos, tales como listas, grafos, etc.
- Implementar más algoritmos para cada esquema algorítmico y completar la aplicación con el esquema de vuelta atrás.
- Realización de las páginas Web de documentación con XML.
- Integrar la aplicación en una página Web a la cual pudieran acceder los alumnos.
- Hacer pruebas masivas con la aplicación para estudiar su utilidad.

Estudiando estas posibilidades, optamos por tratar de cubrir una serie de objetivos que ampliarán la funcionalidad del sistema y, a la vez, facilitarán el uso y la ampliación de la herramienta. Los requisitos funcionales aprobados se enumeran a continuación y se comentan en la siguiente sección:

→ Nuevas estructuras de datos:

Tabla Ordenada:

Permite las operaciones de crear, insertar un elemento, eliminar un elemento, consultar el valor de un elemento, consultar la existencia de un elemento, consultar si la tabla está vacía y recorrer la tabla en inorden.

Tabla Hash:

Permite las operaciones de crear, insertar un elemento, eliminar un elemento, consultar el valor de un elemento, consultar la existencia de un elemento y consultar si la tabla está vacía.

→ Nuevos tipos abstractos de datos:

Consultorio médico:

Se pretende gestionar las consultas de varios médicos y sus pacientes. Permite las operaciones de crear, dar de alta a un médico, pedir consulta, atender consulta, consultar el siguiente paciente de un médico y consultar si un médico tiene pacientes.

Consultorio médico de prioridad:

Pretende reflejar las urgencias en las que, en función de la gravedad, se atiende a los pacientes en el consultorio anterior. Permite las operaciones de crear, dar de alta a un médico, pedir consulta, atender consulta, consultar el siguiente paciente de un médico y consultar si un médico tiene pacientes.

4.3.2. Requisitos Funcionales

Subsistema de Visualización y Animación de la Tabla Ordenada

En cada una de las vistas de la tabla ordenada, podrá verse el comportamiento de esta estructura de datos al aplicar sobre ella las operaciones disponibles. Sobre una tabla ordenada podrán aplicarse las siguientes operaciones:

- Crear
- Insertar
- Eliminar
- Consultar el valor asociado a una clave
- Consultar si está una clave
- Recorrer de manera ordenada respecto a la clave
- Consultar si la tabla está vacía

A continuación veremos con detalle cada una de las operaciones disponibles para el usuario de una tabla ordenada, independientemente de la vista en la que se encuentre.

- ***Crear:*** Al seleccionar esta opción se creará una tabla vacía con dos columnas (clave, valor) sobre la que podrán aplicarse el resto de las operaciones. Previamente se deberá indicar el tipo de las claves y de los valores asociados. Si se selecciona esta opción habiéndose creado una tabla previamente, se empezará desde cero con una nueva estructura.
- ***Insertar:*** Se puede ver cómo se inserta una pareja clave-valor en la tabla (que se ha debido crear previamente).

  Estos valores serán introducidos por el usuario y deben ser del tipo indicado en la creación de la tabla. El valor de la clave que se quiere insertar no debe existir en la tabla. Esto se podría realizar de otras formas diferentes, como por ejemplo, con la combinación del nuevo valor asociado a la clave con el valor antiguo.

  Puede observarse que el tamaño del nodo se adapta al tamaño del texto introducido.

- ***Eliminar:*** Para realizar esta operación será necesario que la tabla ordenada se haya creado y que contenga alguna pareja clave-valor.

  El usuario deberá introducir el valor de la clave que quiere eliminar y en consecuencia se podrá ver como el par formado por esa clave y su valor asociado desaparece de la tabla ordenada.
- ***Consultar un valor asociado a una clave:*** Si se selecciona esta opción, el usuario debe introducir el valor de la clave cuyo valor asociado quiere consultar. La aplicación devuelve dicho valor mostrándolo en la etiqueta descriptiva.

Esta función no provoca cambios en el estado de la tabla ordenada, por lo que el dibujo de la tabla no sufrirá ninguna variación.

Para poder aplicar esta operación será necesario que la tabla ordenada se haya creado y que contenga algún elemento.

o ***Consultar si está una clave:*** Si se selecciona esta opción, el usuario debe introducir el valor de la clave que quiere saber si está. La aplicación devuelve un mensaje con la información en la etiqueta descriptiva.

Esta función no provoca cambios en el estado de la tabla ordenada, por lo que el dibujo de la tabla no sufrirá ninguna variación.

o ***Recorrer ordenada:*** Se realiza el recorrido de la tabla en orden creciente según el valor de la clave. Equivale a hacer un recorrido en inorden del árbol binario de búsqueda que implementa la tabla. Se iluminará cada nodo tratado por el recorrido en el panel gráfico. Para poder realizar esta operación será necesario que la tabla se haya creado con anterioridad.

o ***Consultar si la tabla está vacía***: Si se selecciona esta opción, la aplicación indicará si la tabla ordenada está vacía o si contiene algún elemento, mostrando la información en la etiqueta descriptiva. La tabla se encontrará vacía cuando se cree o cuando se hayan eliminado todos sus elementos. Esta función no provoca cambios en el estado de la tabla, por lo que el dibujo de la misma no sufrirá ninguna variación.

Para poder aplicar esta operación será necesario que la tabla se haya creado con anterioridad.

- **Detalles de la vista usuario de la herramienta**

La representación de la tabla ordenada será mediante una tabla con 2 columnas, una para las claves y otra para los valores. En esta vista no hay limitaciones de tamaño de la tabla. Inicialmente hay 10 filas pero se permite que este tamaño crezca al aumentar el número de elementos de la tabla. En la vista gráfica este efecto queda limitado por el espacio disponible en el panel, que se deberá de subsanar en el futuro con el uso de barras de desplazamiento. Los paneles de dibujo están colocados en vertical.

Explicaremos las características particulares, para esta vista, de algunas de las operaciones.

- o ***Crear***: Al seleccionar esta opción aparecerá una tabla vacía con 10 filas.
- o ***Insertar***: Al insertar un elemento podrá verse cómo se introduce el nuevo par clave, valor en la tabla. El elemento se colocará en orden según la clave. La tabla crecerá en vertical hacia abajo.
- o ***Eliminar***: Cuando se elimine un elemento podrá verse la desaparición de los datos de la fila de la tabla.

- **Detalles de la vista usuario de desarrollo**

Se representa la implementación de la tabla. Los paneles estarán colocados en vertical al igual que en la vista anterior.

Las operaciones aplicables sobre la tabla serán las mismas que las de la tabla general.

La tabla está implementada mediante un árbol binario de búsqueda. Se podrá ver dicho árbol cuyos nodos contienen parejas clave-valor. No habrá limitaciones de tamaño de la tabla, pero gráficamente la visualización no será óptima si el número de elementos es muy grande. En el futuro se deberá arreglar mediante el uso de barras de desplazamiento. La ED crece en vertical hacia abajo.

Las características particulares, para esta vista, de algunas de las operaciones son:

- ***Crear***: Al seleccionar esta opción aparecerá un árbol vacío representado como una toma de tierra.
- ***Insertar***: Al insertar un elemento podrá verse cómo se introduce un nuevo nodo en el árbol.
- ***Eliminar***: Cuando se elimine un elemento podrá verse cómo desaparece el nodo del árbol. Si la estructura arbórea sólo consta del elemento que se va a suprimir, el elemento se borrará y se mostrará el árbol vacío.

Si el árbol consta de más de un nodo, ese elemento se eliminará y el resto de los nodos se reordenarán animadamente, manteniendo la relación de orden.

Si el nodo a eliminar tiene hijo derecho e izquierdo, o si sólo tiene hijo derecho, colocaremos en la posición que ocupaba el elemento, el menor elemento del hijo derecho. Si sólo tiene hijo izquierdo colocaremos éste en la posición del elemento a eliminar. Para visualizar mejor el intercambio se mostrará una flecha indicando el elemento que va a sustituir al anterior, y el elemento a eliminar se mostrará tachado con un aspa.

Subsistema de Visualización y Animación de la Tabla Hash

En cada una de las vistas de la tabla hash, podrá verse el comportamiento de esta estructura de datos al aplicar sobre ella las operaciones disponibles. Sobre una tabla hash podrán aplicarse las siguientes operaciones:

- Crear
- Insertar
- Eliminar
- Consultar el valor asociado a una clave
- Consultar si está una clave
- Consultar si la tabla está vacía

A continuación veremos con detalle cada una de las operaciones disponibles para el usuario de una tabla hash, independientemente de la vista en la que se encuentre.

- ***Crear:*** Al seleccionar esta opción se creará una tabla vacía con dos columnas (clave, valor) sobre la que podrán aplicarse el resto de las operaciones. Previamente se deberá indicar el tipo de las claves y de los valores asociados. Si se selecciona esta opción habiéndose creado una tabla previamente, se empezará desde cero con una nueva estructura.
- ***Insertar:*** Se puede ver cómo se inserta una pareja clave-valor en la tabla (que se ha debido crear previamente).

  Estos valores serán introducidos por el usuario y deben ser del tipo indicado en la creación de la tabla. La clave puede existir en la tabla, en cuyo caso se actualizará el valor asociado con el nuevo valor introducido. Se puede comprobar que se ha resuelto de forma diferente a como se resolvió para las tablas ordenadas. Esto se debe a nuestra intención de mostrar a los alumnos los diferentes modos de realizar esta operación.
- ***Eliminar:*** Para realizar esta operación será necesario que la tabla hash se haya creado y que contenga alguna pareja clave-valor.

  El usuario deberá introducir el valor de la clave que quiere eliminar y en consecuencia se podrá ver como el par formado por esa clave y su valor asociado desaparece de la tabla hash si dicha clave existe. En caso de no estar la clave en la tabla se informará de este hecho al usuario.
- ***Consultar un valor asociado a una clave:*** Si se selecciona esta opción, el usuario debe introducir el valor de la clave cuyo valor asociado quiere consultar. La aplicación devuelve dicho valor mostrándolo en la etiqueta descriptiva.

  Esta función no provoca cambios en el estado de la tabla hash, por lo que el dibujo de la tabla no sufrirá ninguna variación.

  Para poder aplicar esta operación será necesario que la tabla hash se haya creado y que contenga algún elemento (se informará al usuario en caso contrario).
- ***Consultar si está una clave:*** Si se selecciona esta opción, el usuario debe introducir el valor de la clave que quiere saber si está. La aplicación devuelve un mensaje con la información en la etiqueta descriptiva.

  Esta función no provoca cambios en el estado de la tabla hash, por lo que el dibujo de la tabla no sufrirá ninguna variación.
- ***Consultar si la tabla está vacía***: Si se selecciona esta opción, la aplicación indicará si la tabla hash está vacía o si contiene algún elemento, mostrando la solución en la etiqueta descriptiva. La tabla se encontrará vacía cuando se cree o cuando se hayan eliminado todos sus elementos. Esta función no provoca cambios en el estado de la tabla, por lo que el dibujo de la misma no sufrirá ninguna variación.

Para poder aplicar esta operación será necesario que la tabla se haya creado con anterioridad.

- **Detalles de la vista de usuario de la herramienta**

La representación de la tabla hash será mediante una tabla con 2 columnas, una para las claves y otra para los valores. En esta implementación no habrá limitaciones de tamaño de la tabla. Inicialmente hay 10 filas pero cuando se sobrepase la mitad del tamaño de la tabla éste se duplica. Los paneles de dibujo están colocados en vertical.

Explicaremos las características particulares, para esta vista, de algunas de las operaciones.

- ***Crear***: Al seleccionar esta opción aparecerá una tabla vacía con 10 filas.
- ***Insertar***: Al insertar un elemento podrá verse cómo se introduce el nuevo par clave, valor en la tabla. El elemento se colocará en una posición cualquiera de la tabla, sin seguir un orden.

- **Detalles de la vista de usuario de desarrollo**

Se representa cómo está implementada la tabla. Los paneles estarán colocados en vertical al igual que en la vista anterior.

Las operaciones aplicables sobre la tabla serán las mismas que las de la tabla general.

Implementación con tabla hash abierta

Se podrá ver una tabla con una única columna de valores hash y una lista enlazada de parejas clave-valor asociada a cada valor hash. No hay limitación en cuanto al tamaño de la tabla. Se mostrará información adicional que indique la operación que se esta realizando, así como la función hash utilizada y los datos introducidos por el usuario.

Implementación con tabla hash cerrada

Se podrá ver una tabla con dos columnas, una para claves y otra para valores, al igual que la vista de usuario de la herramienta. En este caso, se debe mostrar información adicional como la función hash utilizada, el valor hash asociado a cada clave, la función de rehashing que se utiliza en caso de existir una colisión entre claves, y el valor hash asociado a cada clave en cada intento. También se mostrará si existen colisiones al insertar, o si existe la posibilidad de adelantar la clave en la tabla. Para esto, se mostrarán en color verde las posiciones de la tabla que estaban ocupadas pero que han sido borradas y se pueden reutilizar.

Subsistema de Visualización y Animación del TAD Consultorio médico

En ambos consultorios, tanto el normal como el de prioridad, se permiten las mismas operaciones. En cada una de las vistas del consultorio, podrá verse el comportamiento de este tipo abstracto de datos al aplicar sobre el las operaciones disponibles. Sobre un consultorio podrán aplicarse las siguientes operaciones:

- Crear
- Dar de alta un nuevo médico
- Pedir cita con un médico
- Atender a un paciente
- Consultar el siguiente paciente de un médico
- Consultar si tiene pacientes un médico

A continuación veremos con detalle cada una de las operaciones disponibles para el usuario de un consultorio, independientemente de la vista en la que se encuentre.

- ***Crear:*** al seleccionar esta opción se creará un consultorio vacío. Previamente se deberá indicar el tipo de los médicos y de los pacientes asociados. Si se selecciona esta opción habiéndose creado un consultorio previamente, se empezará desde cero con un nuevo consultorio.
- ***Dar de alta un nuevo médico:*** se puede ver cómo se añade un médico con un identificador en el consultorio (que se ha debido crear previamente).

  Este identificador será introducido por el usuario y debe ser del tipo indicado en la creación del consultorio. El identificador no puede existir en el consultorio, por lo que se mostrará un mensaje de error en el caso de introducir un médico existente.
- ***Pedir cita con un médico:*** para realizar esta operación será necesario que el consultorio esté creado y contenga algún médico.

  El usuario deberá introducir el médico con el que quiere pedir cita y el identificador asociado al paciente. En caso de no existir el médico se informará al usuario.

  Se visualiza la inserción de un nuevo paciente con el médico elegido.
- ***Atender a un paciente:*** para realizar esta operación será necesario que el consultorio se haya creado y que contenga algún paciente.

  El usuario deberá introducir el médico que quiere que atienda a su primer paciente. Se podrá ver como desaparece del consultorio el primer paciente del médico elegido en caso de existir. En otro caso se informará al usuario de la inexistencia de ese médico.
- ***Consultar el siguiente paciente de un médico:*** si se selecciona esta opción, el usuario debe introducir el valor del médico que quiere consultar. La aplicación devuelve su primer paciente mostrándolo en la etiqueta descriptiva.

Esta función no provoca cambios en el estado del consultorio, por lo que el dibujo del consultorio no sufrirá ninguna variación.

Para poder aplicar esta operación será necesario que el consultorio se haya creado y que contenga algún elemento (se informará al usuario en caso contrario).

o ***Consultar si tiene pacientes un médico:*** si se selecciona esta opción, el usuario debe introducir el valor del médico que quiere consultar. La aplicación devuelve si el médico tiene pacientes, mostrándolo en la etiqueta descriptiva.

Esta función no provoca cambios en el estado del consultorio, por lo que el dibujo del consultorio no sufrirá ninguna variación.

Para poder aplicar esta operación será necesario que el consultorio se haya creado (se informará al usuario en caso contrario).

En el consultorio médico sólo se permitirá el alta a 3 médicos por limitaciones de visualización. La implementación no tiene ese problema y con el uso de barras de desplazamiento se podría aumentar el número de médicos del consultorio.

- **Detalles de la vista de usuario de la herramienta**

La representación del consultorio será mediante tres colas, correspondiendo cada una a un médico.

Consultorio básico

Se representa al médico con la puerta de su consulta y la cola de pacientes con iconos de personas dispuestos en el orden de creación.

Consultorio de prioridad

Se representa al médico con un icono de médico, se distingue al paciente más prioritario y tras una puerta se representa al resto de pacientes.

- **Detalles de la vista de implementación**

Se representa cómo está implementado el consultorio. La vista coincide con la representación de una tabla hash cerrada, puesto que el consultorio se construye con ella.

Consultorio básico

Se representa al médico como una clave y sus pacientes se muestran como una cola implementada dinámicamente.

Consultorio de prioridad

Se representa al médico como una clave y sus pacientes se muestran como un array estático de diez posiciones que representa un montículo de mínimos.

Las operaciones aplicables sobre este consultorio serán las mismas que las del consultorio básico.

4.3.3. Requisitos no funcionales

Requisitos del sistema

- Disponer de la Máquina Virtual de Java, JDK 1.4.2.
- Intel Pentium II (AMD K6) o superior.
- Al menos 10 MB. de espacio de disco duro.

Requisitos de diseño

- Es importante construir un sistema que sea modular para poder realizar pruebas periódicas sin alterar el desarrollo de la aplicación. Esto se debe a que cada estructura de datos y cada esquema algorítmico es independiente y se debe poder probar con facilidad sin integrar con el resto del sistema para poder encontrar defectos o errores.
- De la misma forma debe mantener esta modularidad para permitir una fácil ampliación que proporcione nuevas funcionalidades a la herramienta.
- La interfaz de usuario debe mantener el estilo y la apariencia en sus nuevas ampliaciones para facilitar el entendimiento de la misma por parte del usuario.

La estructura modular de la herramienta y los pasos a seguir para poder ampliarla correctamente serán explicados extensamente en la sección 5.

4.4. Glosario

4.4.1. Definiciones

- **Tabla ordenada:** estructura de datos que asocia claves con valores y los coloca de forma ordenada por claves.
- **Tabla hash abierta:** estructura de datos que asocia claves con valores agrupadas entorno al valor hash.
- **Tabla hash cerrada:** estructura de datos que asocia claves con valores realizando un rehash en caso de producirse una colisión entre claves.
- **Consultorio médico:** tipo abstracto de datos que simula la llegada y atención de pacientes a un médico, siendo la atención por orden de llegada.
- **Consultorio médico de prioridad:** tipo abstracto de datos que simula la llegada y atención de pacientes a un médico, siendo la atención en función de la prioridad de cada paciente.
- **Tipo abstracto de datos:** modelo matemático caracterizado por una colección de operaciones sobre un conjunto de datos.

4.4.2. Acrónimos

- EDs: Estructuras de datos.
- ED: Estructura de datos.
- TAD: Tipo abstracto de datos.
- GUI: Interfaz gráfica de usuario.
- EVS: Estudio de Viabilidad del Sistema.
- ABB: Árbol binario de búsqueda.
- AVL: Son las iniciales de Adelson-Velskii y Landis, los cuales idearon este tipo de árbol.
- LDC: Líneas De Código.

5.1. Modularización de la interfaz gráfica de usuario.

Uno de los aspectos más importantes a tener en cuenta a la hora de diseñar o ampliar una aplicación es que el código esté organizado lo más modularmente posible. Así, se gana claridad de forma que la comprensión del código por parte del programador y la ampliación y/o modificación del mismo resultan mucho más sencillas.

Esto se tuvo muy en cuenta en el proyecto original de forma que está perfectamente separada la implementación de las EDs y algoritmos de la parte gráfica de la herramienta. El diseño constaba de tres módulos principales, la interfaz, el paquete gráfico y el paquete de implementación.

Aun así, a la hora de estudiar en profundidad el código para introducir nuevo conocimiento nos dimos cuenta que se podía llegar a conseguir una mayor modularización. A continuación se explica los motivos que nos hicieron tomar la decisión de dar prioridad a este hecho y los pasos que seguimos para conseguirlo.

5.1.1. ¿Por qué decidimos modularizar?

Una de las mayores dificultades con la que nos encontramos a la hora de ampliar la herramienta fue la comprensión de la clase que implementaba la interfaz gráfica con el usuario (*Interfaz.java*).
Dicha clase era la responsable tanto de crear las interfaces gráficas de las EDs y de los algoritmos como de actualizarla según las acciones realizadas por el usuario. Pertenecía al paquete *principal.* Este paquete contenía también las siguientes clases:

- *BarraDesplazamiento.java:* clase encargada de calcular la posición de inicio donde pintar el dibujo. También determina el tamaño y situación de la barra de desplazamiento.
- *PanelDibujo.java:* clase encargada de llamar al método que implementa la operación pintar de cada ED y método algorítmico.
- *VentanaInicio.java:* Se encarga de dar la bienvenida al usuario y explicar el objetivo de la herramienta.
- *VentanaSelección.java*: Muestra la ventana selección que permite al usuario seleccionar la ED o esquema algorítmico con el que desea trabajar.

La clase *Interfaz* constaba de:

- 8000 líneas de código.
- 81 métodos.
- 184 atributos privados.

A la vista de estos datos la clase interfaz era inmanejable. Este fue uno de los principales motivos que nos llevó a dar prioridad a esta idea, ya que en el futuro en esta clase se seguirían aumentando las líneas de código y resultaría bastante difícil a los futuros implementadores de la herramienta tratar con ella.

Cada vez que se quería añadir una nueva ED o un nuevo algoritmo era necesario inspeccionar las 8000 líneas para añadir el código necesario, ya que los métodos son compartidos por la mayoría de las EDs y de los algoritmos llevando a cabo una distinción de casos entre ellos. Esta clase seguiría por tanto aumentando su tamaño, agravando este problema.

Hay que añadir que la clase *Interfaz* es un elemento que puede pertenecer a un grupo (paquete) distinto, ya que su funcionalidad no tiene relación con el resto de las clases incluidas en el paquete *principal*.

5.1.2. Ideas previas

Para llevar a cabo el proceso de modularización se pensó en dos posibilidades:

- Diseñar dos clases separadas: una para la interfaz gráfica de las ED's y otra para la de los algoritmos.
- Diseñar clases separadas para cada estructura de datos y para cada algoritmo.

Se eligió la segunda opción porque de esta forma la aplicación es más modular, con las ventajas que esto conlleva. Por ejemplo, cada ED y cada algoritmo sólo conocen sus propios métodos. También, al introducir una nueva ED o un nuevo algoritmo, se tendrá una clase separada del resto para su interfaz gráfica. Así no se influye en las ED o algoritmos ya implementados. Lo que ya estaba hecho y se había probado que funcionaba no se modifica. Por tanto añadir conocimiento a la aplicación no puede estropear lo ya existente.

Por otro lado para la implementación de cada clase se pensó en dos ideas:

- Que cada clase contuviese sus propios métodos, es decir los métodos que necesita.
- Los métodos comunes a la mayoría de las EDs y algoritmos estuvieran en una clase común, de la cual heredarían las clases específicas.

Por ejemplo, en el contexto de las estructuras de datos, los principales métodos que tienen comunes son los siguientes:

- void jMenuFuncCrear_actionPerformed(ActionEvent e): método asociado a la función crear de la ED.
- void nuevoElemento (): encargado de añadir un nuevo elemento en la ED.
- void jMenuFuncQuitarElemento_actionPerformed(ActionEvent e): método asociado a la función eliminar de la ED.
- void jMenuFuncVacia_actionPerformed(ActionEvent e): método asociado a la función que consulta si la estructura de datos está vacía.
- void jMenuFuncEsta_actionPerformed(ActionEvent e): método asociado a la función que consulta si un elemento está en la estructura de datos.
- void jbInit() : Método que se encarga de construir la ventana sobre la que trabajará el usuario.

Se eligió la segunda opción, ya que la primera no suponía una ventaja para la modularidad. Al hacer una clase con métodos comunes para la mayoría de EDs y algoritmos, en los métodos habría que distinguir casos para cada uno (9 casos para las EDs y 5 para los algoritmos) y esto era lo que se hacía originalmente. Al implementar el mismo método acciones distintas para cada ED o algoritmo, al introducir una nueva estructura o problema, se podría estropear lo ya existente.
Sin embargo, al tener cada clase sus propios métodos, al ampliar la herramienta las clases existentes no se ven modificadas y pueden servir de guía para la introducción de la nueva ED o problema. El único inconveniente es que la cantidad total de código puede crecer.

Para mayor claridad, se muestra a continuación un ejemplo de cómo sería un método compartido por todas las EDs en una clase común y cómo sería si cada una implementara el suyo propio.

Primero vemos parte del código del método crear compartido por todas las EDs. Se resaltan las líneas donde se distinguen casos.

```
void jMenuFuncCrear_actionPerformed(ActionEvent e)
 {
      //Definición del mensaje de la etiqueta descriptiva
      if(nomb.equals("Arbol") || nomb.equals("AVL"))
        jLabel1.setText("Se ha creado el " +  nomb + " de " + tipoED.toString());
      else
          jLabel1.setText("Se ha creado la " + nomb + " de " + tipoED.toString());

      //Implementación de pilas basadas en array
      if(nomb.equals("Pila"))
        pila = new PilasArray();
      //Implementación de colas basadas en array
      if(nomb.equals("Cola"))
        cola = new ColaArrayCircular();
      //Implementación árbol BB
      if(nomb.equals("Arbol"))
        arbolBB = new ABB();
          .
          .
          .
          .
  //Se construye los paneles de dibujo para cada vista (usuario, estática y dinámica)
      if(nomb.equals("Pila"))
      {
        panelDibujo = new PanelDibujo(panelGrafico,null, nomb, pila,"herramienta");
        panelDibujo2 = new PanelDibujo(panelGrafico2,null, nomb, pila,"estatica");
        panelDibujo3 = new PanelDibujo(panelGrafico3,null, nomb, pila,"dinamica");
      }

      if(nomb.equals("Cola"))
      {
        panelDibujo = new PanelDibujo(panelGrafico,null, nomb, cola,"herramienta");
        panelDibujo2 = new PanelDibujo(panelGrafico2,null, nomb, cola,"estatica");
        panelDibujo3 = new PanelDibujo(panelGrafico3,null, nomb, cola,"dinamica");
      }
      if(nomb.equals("Arbol"))
      {
          panelDibujo=new PanelDibujo(panelGrafico,null, nomb, arbolBB,"herramienta");
          panelDibujo2 = new PanelDibujo(panelGrafico2,null, nomb, arbolBB,"estatica");
          panelDibujo3 = new PanelDibujo(panelGrafico3,null, nomb,
          arbolBB,"dinamica");
      }
          .
          .
          .
          .
    }
```

Sin embargo, si cada clase implementa su propio método crear, su código sería el siguiente (ejemplo para la clase GUIPila que implementa los métodos necesarios para crear y actualizar la interfaz gráfica de usuario para la ED Pila):

```
void jMenuFuncCrear_actionPerformed(ActionEvent e)
 {
          jLabel1.setText("Se ha creado la pila de " + tipoED.toString());

      //Implementación de pilas basadas en array
        pila = new PilasArray();

          .
          .
          .
          .

  //Se construye los paneles de dibujo para cada vista (usuario, estática y dinámica)
        panelDibujo = new PanelDibujo(panelGrafico,null, nomb, pila,"herramienta");
        panelDibujo2 = new PanelDibujo(panelGrafico2,null, nomb, pila,"estatica");
        panelDibujo3 = new PanelDibujo(panelGrafico3,null, nomb, pila,"dinamica");

          .
          .
          .
          .

    }
```

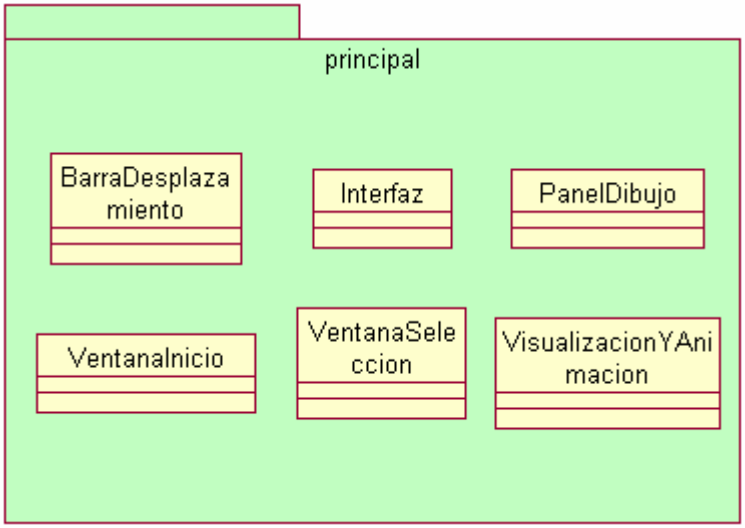
Cada clase implementa su propio método. Como se puede apreciar, no es necesario distinguir casos, haciendo el código más sencillo, más manejable y con un mayor grado de reusabilidad.

5.1.3. Proceso de modularización

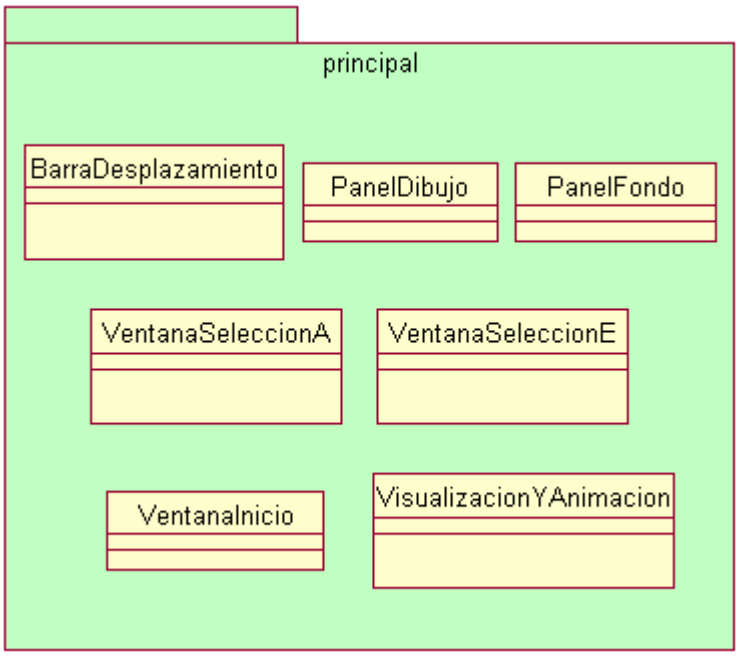
- **Cambios en la estructura de implementación del proyecto:**

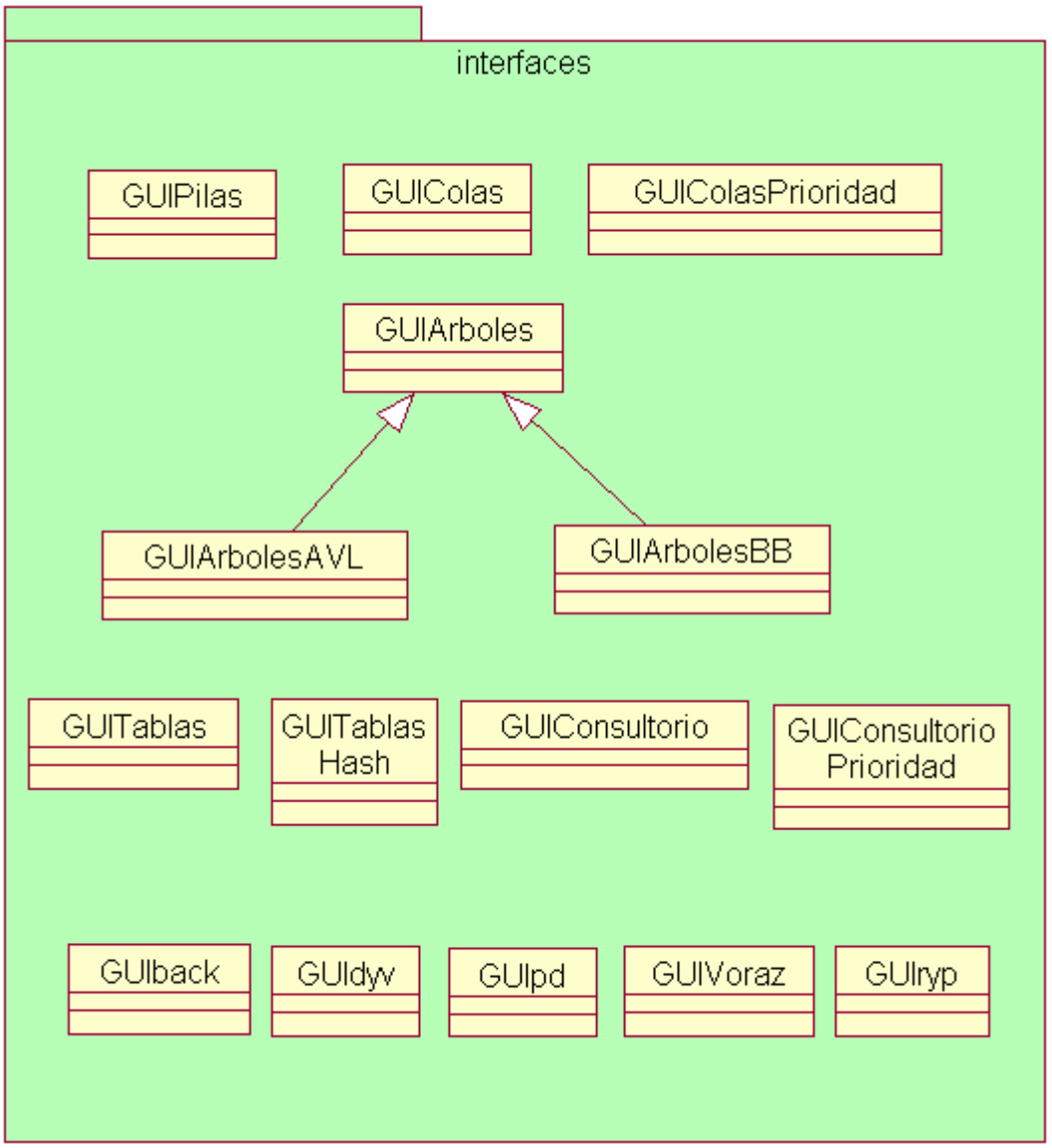
Creamos un nuevo paquete llamado “interfaces” para separar las interfaces gráficas del resto de funcionalidades. En este paquete se creó una clase para cada tipo de estructura de datos y para cada tipo de algoritmo.

Por lo tanto se pasa de tener el paquete principal con las siguientes clases:



A tener el mencionado paquete interfaces y el paquete principal modificado como sigue:





Como se puede observar, la clase “Interfaz” ya no existe. Se ha modularizado dando lugar a las clases contenidas en el paquete “interfaces”. Como aclaración hay que decir que la clase PanelFondo (incluida en el paquete *principal*) es la encargada de establecer una imagen como fondo en un panel.

- **Métodos propios de cada ED y cada algoritmo:**

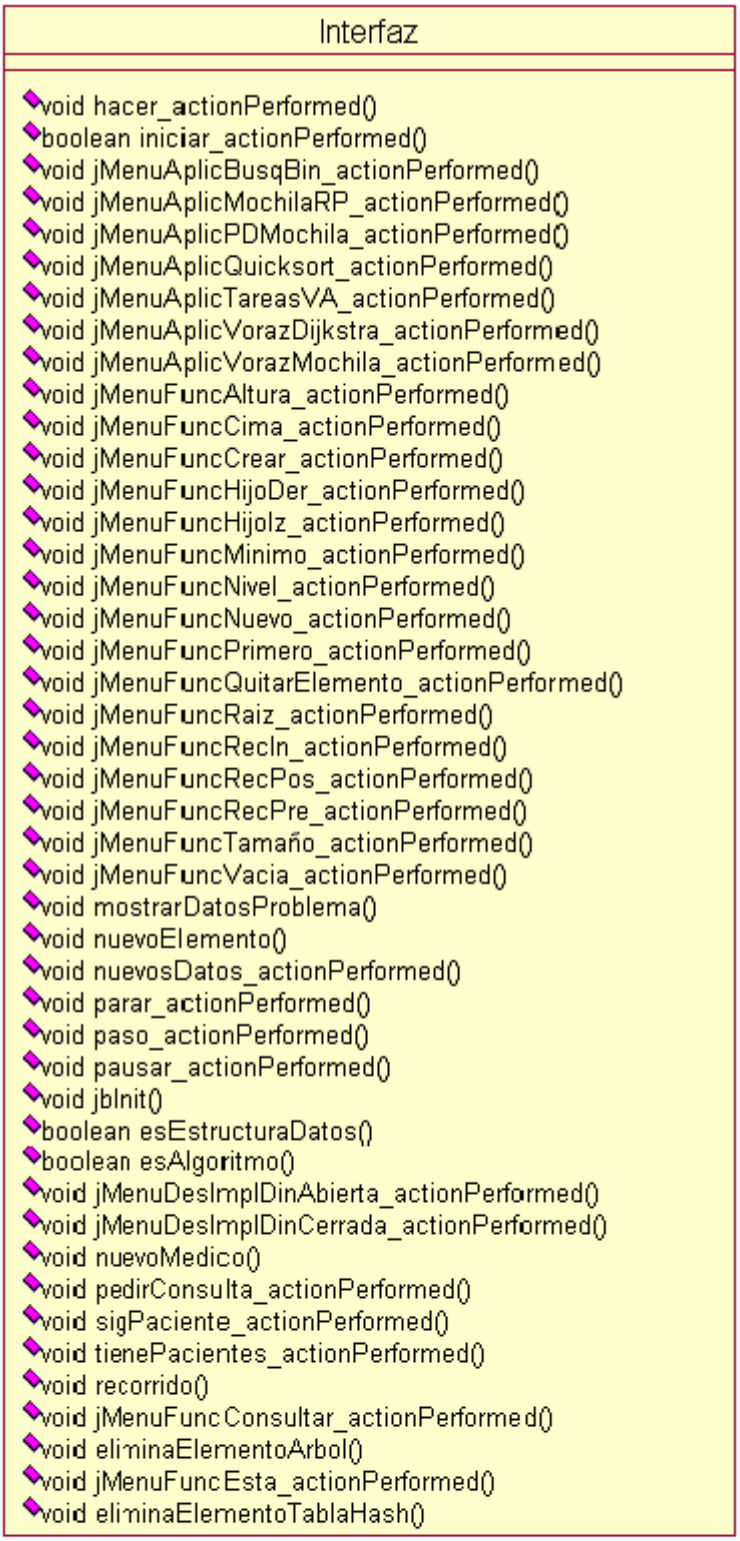
En la misma clase (*Interfaz.java)* estaban los métodos de las estructuras de datos y de los algoritmos.

Distintas estructuras de datos compartían atributos tales como botones, checkbox, etc, por lo que también era necesario distinguir casos (al igual que se ha explicado para el método *jbInit*) en los métodos *actionPerformed().*

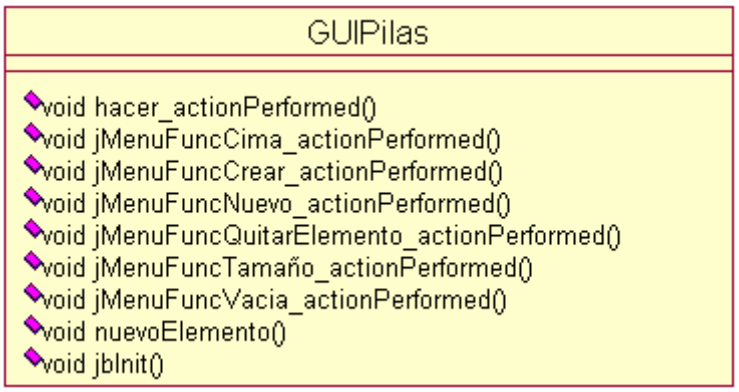
A la hora de saber las funciones de cada ED o algoritmo había que estudiar con detalle cada método y cada atributo para ver el *actionPerformed()* al que se asociaban.

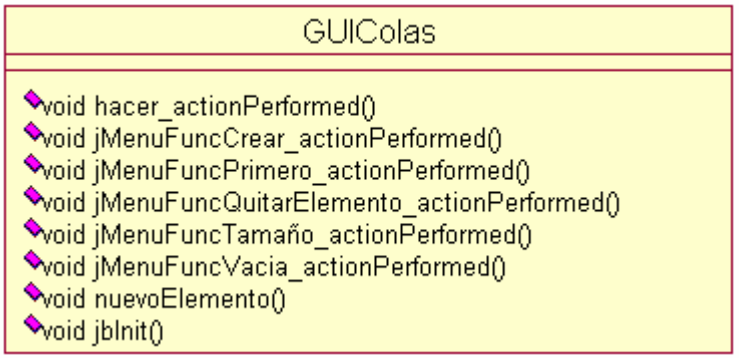
Después de modularizar, cada *actionPerformed()* se ocupará de las acciones de la estructura de datos o del algoritmo a cuya clase pertenezca.

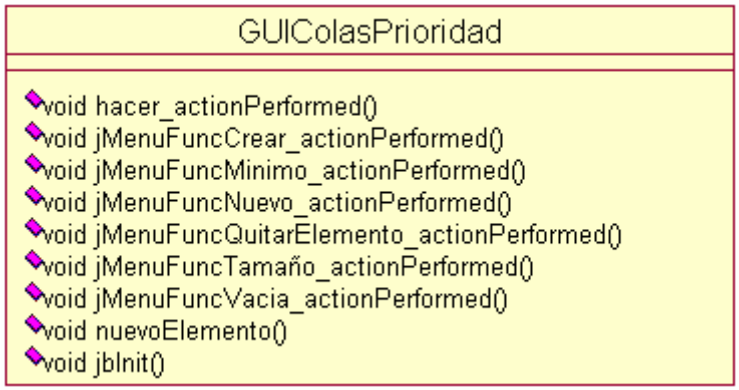
Se muestran los métodos principales de la clase *Interfaz.* Como se puede apreciar en el siguiente diagrama de clases, contiene todos los métodos de todas las EDs y algoritmos necesarios para la interfaz gráfica con el usuario:

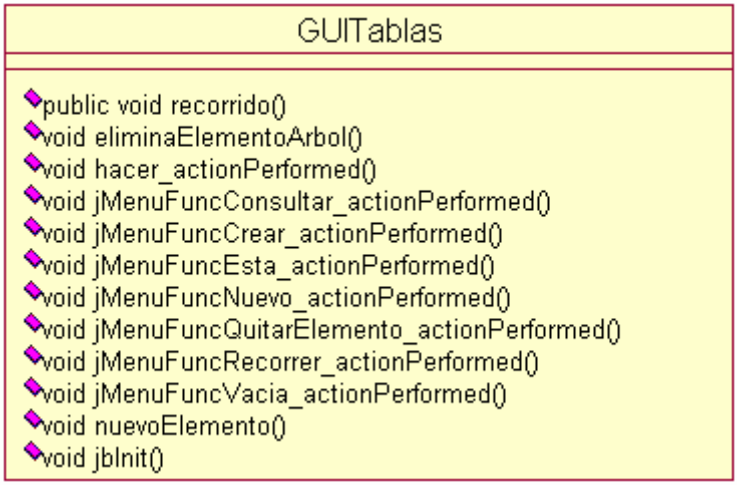

Interfaz
void hacer_actionPerformed()
boolean iniciar_actionPerformed()
void jMenuAplicBusqBin_actionPerformed()
void jMenuAplicMochilaRP_actionPerformed()
void jMenuAplicPDMochila_actionPerformed()
void jMenuAplicQuicksort_actionPerformed()
void jMenuAplicTareasVA_actionPerformed()
void jMenuAplicVorazDijkstra_actionPerformed()
void jMenuAplicVorazMochila_actionPerformed()
void jMenuFuncAltura_actionPerformed()
void jMenuFuncCima_actionPerformed()
void jMenuFuncCrear_actionPerformed()
void jMenuFuncHijoDer_actionPerformed()
void jMenuFuncHijoIz_actionPerformed()
void jMenuFuncMinimo_actionPerformed()
void jMenuFuncNivel_actionPerformed()
void jMenuFuncNuevo_actionPerformed()
void jMenuFuncPrimero_actionPerformed()
void jMenuFuncQuitarElemento_actionPerformed()
void jMenuFuncRaiz_actionPerformed()
void jMenuFuncRecIn_actionPerformed()
void jMenuFuncRecPos_actionPerformed()
void jMenuFuncRecPre_actionPerformed()
void jMenuFuncTamaño_actionPerformed()
void jMenuFuncVacia_actionPerformed()
void mostrarDatosProblema()
void nuevoElemento()
void nuevosDatos_actionPerformed()
void parar_actionPerformed()
void paso_actionPerformed()
void pausar_actionPerformed()
void jbInit()
boolean esEstructuraDatos()
boolean esAlgoritmo()
void jMenuDeslmplDinAbierta_actionPerformed()
void jMenuDeslmplDinCerrada_actionPerformed()
void nuevoMedico()
void pedirConsulta_actionPerformed()
void sigPaciente_actionPerformed()
void tienePacientes_actionPerformed()
void recorrido()
void jMenuFuncConsultar_actionPerformed()
void eliminaElementoArbol()
void jMenuFuncEsta_actionPerformed()
void eliminaElementoTablaHash()

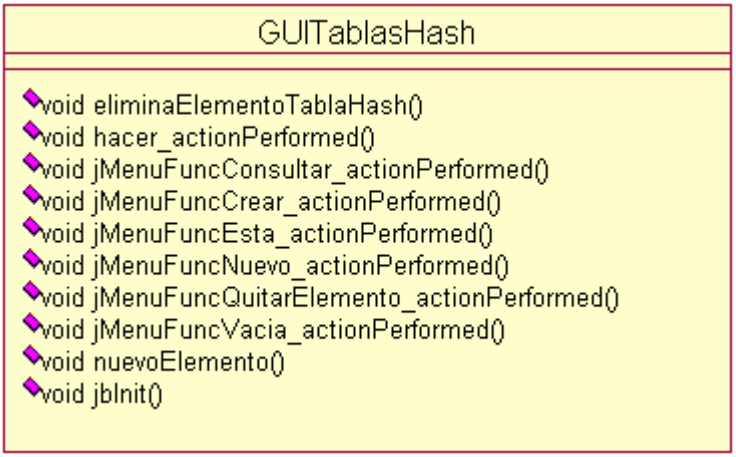
Una vez modularizado, se tienen las clases siguientes para las estructuras de datos:

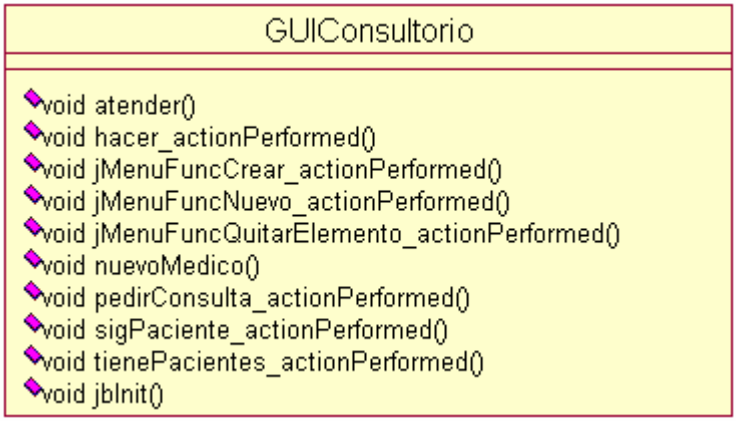


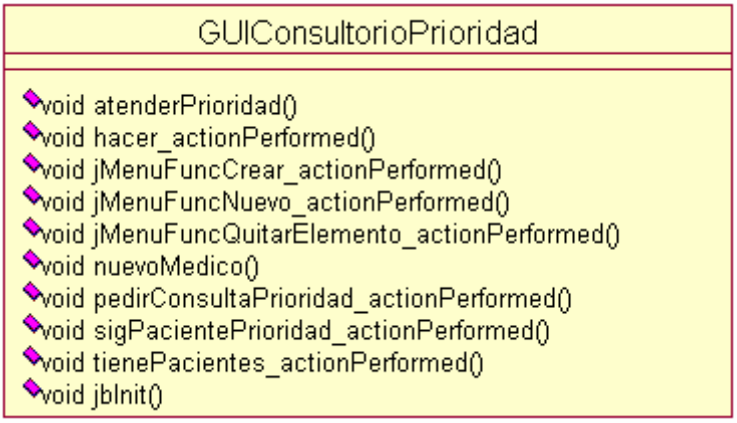




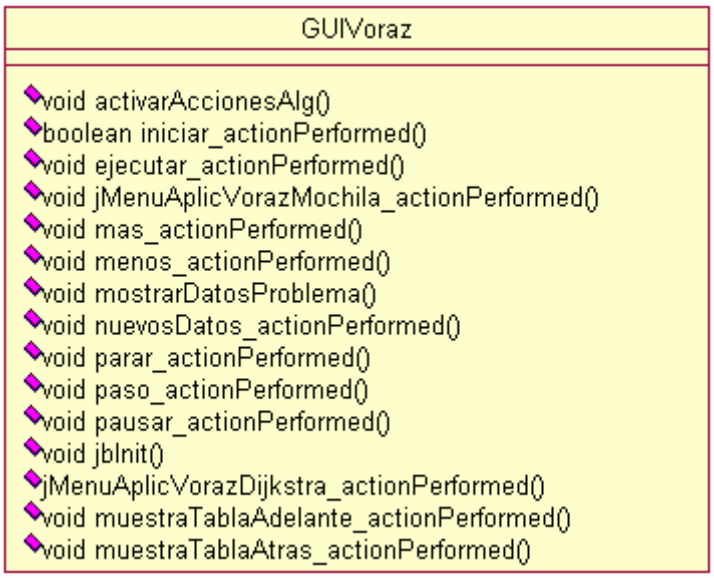

GUITablas
public void recorrido()
void eliminaElementoArbol()
void hacer_actionPerformed()
void jMenuFuncConsultar_actionPerformed()
void jMenuFuncCrear_actionPerformed()
void jMenuFuncEsta_actionPerformed()
void jMenuFuncNuevo_actionPerformed()
void jMenuFuncQuitarElemento_actionPerformed()
void jMenuFuncRecorrer_actionPerformed()
void jMenuFuncVacia_actionPerformed()
void nuevoElemento()
void jbInit()

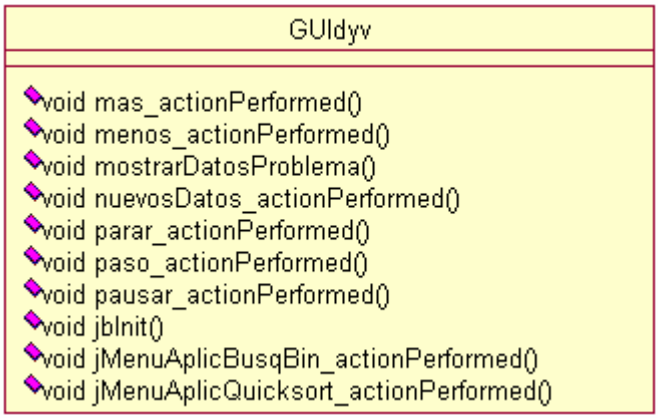

GUITablasHash
void eliminaElementoTablaHash()
void hacer_actionPerformed()
void jMenuFuncConsultar_actionPerformed()
void jMenuFuncCrear_actionPerformed()
void jMenuFuncEsta_actionPerformed()
void jMenuFuncNuevo_actionPerformed()
void jMenuFuncQuitarElemento_actionPerformed()
void jMenuFuncVacia_actionPerformed()
void nuevoElemento()
void jbInit()

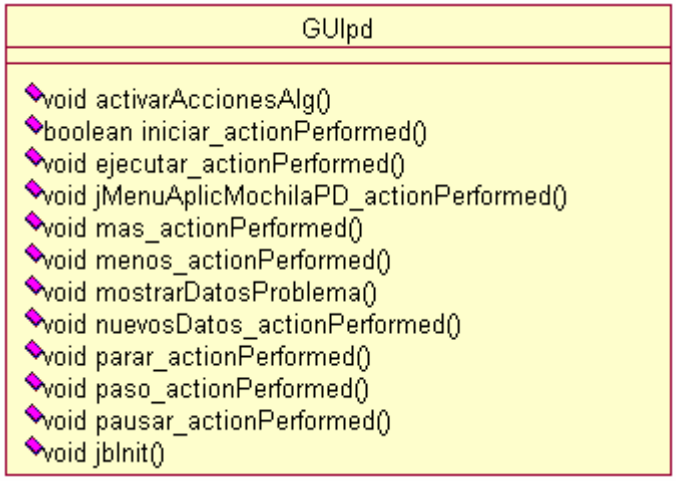

GUIConsultorio
void atender()
void hacer_actionPerformed()
void jMenuFuncCrear_actionPerformed()
void jMenuFuncNuevo_actionPerformed()
void jMenuFuncQuitarElemento_actionPerformed()
void nuevoMedico()
void pedirConsulta_actionPerformed()
void sigPaciente_actionPerformed()
void tienePacientes_actionPerformed()
void jbInit()

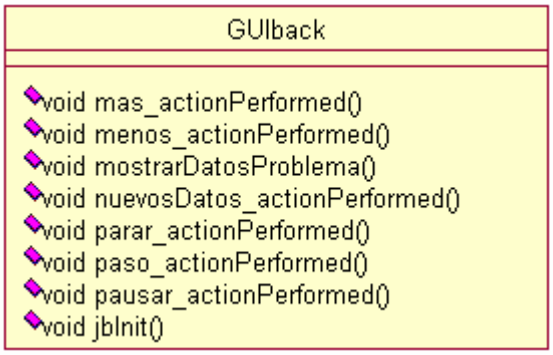


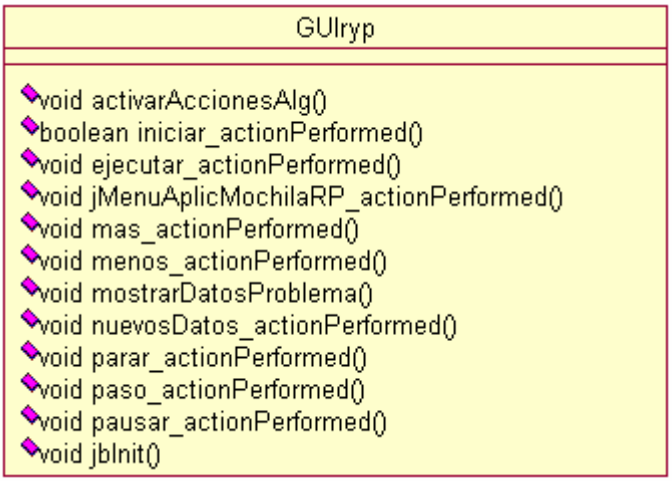
Y las clases que siguen para los algoritmos:









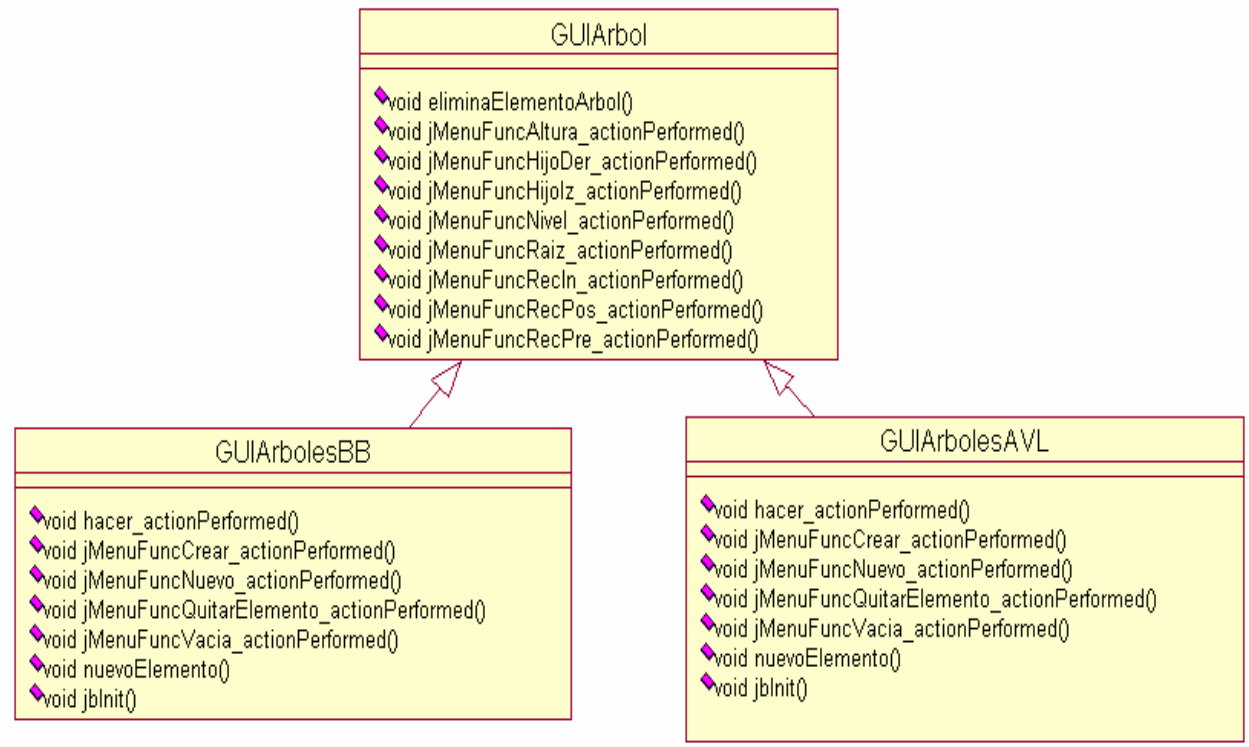


- **Caso de los árboles:**

  En la aplicación están implementados dos tipos de árboles: los árboles binarios de búsqueda y los árboles AVL.

  A la hora de modularizarlos, nos dimos cuenta de que tenían muchas cosas en común. Los métodos propios de árboles tales como los que calculan la altura, el nivel, el hijo derecho, el izquierdo, la raíz, etc. eran prácticamente iguales en los dos, salvo por algún detalle por lo que se distinguían casos.

Pensando en ello decidimos que en este caso sí se podía ganar bastante si estos métodos específicos se heredaban de una clase padre.
Lo que se tiene es lo siguiente:



Las clases que permanecen en los tipos específicos de árboles son las que implementan las acciones de crear, insertar un elemento, consultar si está vacía, etc. que puede realizar el usuario en cada interfaz gráfica.
Como ya se ha mencionado, las clases incluidas en la clase padre, son las propias de los árboles que puede realizar el usuario.

- **Método que crea la ventana inicial:**

  En la clase Interfaz había un método *jbInit()* que era el responsable de crear la interfaz gráfica inicial. En este método se creaban todos los componentes necesarios para cada estructura y algoritmo y no era trivial la adición de una nueva ED o algoritmo. Había que tener bastante cuidado a la hora de cambiar las condiciones de las distinciones de casos ya que se podía estropear lo ya implementado. El siguiente fragmento de código muestra este problema, hay distinciones de casos para saber si se trabaja con una ED o con un algoritmo y dentro de cada uno, para cada tipo de ED y para cada algoritmo. Por ello, para ampliar la herramienta, hay que estudiar a fondo las funciones y los elementos comunes que tiene la nueva ED o el nuevo problema con lo que hay ya implementado, para saber a que distinción de casos se corresponde.

```
void jbInit() throws Exception  {

        if (esEstructuraDatos(nomb))
           {
            //operación de añadir un nuevo elemento
            if (nomb.equals("Pila"))
            {
              nombAccion = "Apilar";
              comAccion = "Apilar";
            }

            if (nomb.equals("Cola"))
            {
              nombAccion = "Añadir";
              comAccion = "Añadir";
            }

            if (nomb.equals("Arbol") || nomb.equals("AVL") ||
                nomb.equals("Cola de Prioridad"))
            {
              nombAccion = "Insertar";
              comAccion = "Insertar";
            }

        jMenuFunc1.setActionCommand(comAccion);
            jMenuFunc1.setText(nombAccion);
            jMenuFunc1.setEnabled(false);
            jMenuFunc1.addActionListener(new java.awt.event.ActionListener() {
              public void actionPerformed(ActionEvent e) {
                jMenuFuncNuevo_actionPerformed(e);
              }
            });

        //lo mismo se haría para el resto de funciones (eliminar, está, vacía)
                .
                .
                .
```

```
        //operación específica de los árboles
                if (nomb.equals("Arbol") || nomb.equals("AVL"))
                    {

                     jMenuFunc7.setActionCommand("Nivel");
                     jMenuFunc7.setText("Nivel");
                     jMenuFunc7.setEnabled(false);
                     jMenuFunc7.addActionListener(new java.awt.event.ActionListener() {
                       public void actionPerformed(ActionEvent e) {
                         jMenuFuncNivel_actionPerformed(e);
                       }
                     });
            }
        }

          else
           {
            //Menú Aplicación
            jMenuAplic.setActionCommand("Aplic");
            jMenuAplic.setText("Aplicaciones");


         if (nomb.equals("Voraz") || nomb.equals("PD") ||
            nomb.equals("RP"))
            {
              nombAccion = "Problema de la mochila";
              comAccion = "Mochila";
            }

            if (nomb.equals("DV"))
            {
              nombAccion = "Quicksort";
              comAccion = "Quicksort";
            }

            if (nomb.equals("VA"))
            {
              nombAccion = "Tareas con beneficio";
              comAccion = "Tareas";
            }

                  .
                  .
                  .
    }
```

Después de modularizar, cada clase tiene su propio método *jbInit()* con los atributos que necesita.

A continuación se muestra un fragmento de código de dicho método para la ED Pila. Como se puede observar, ahora este método sólo contiene las funciones de las pilas, sin necesidad de distinguir casos.
Así mismo los componentes que se crean, son sólo los que necesitan las pilas.

Lo mismo ocurriría para el resto de las clases.

```
        void jbInit() throws Exception  {

                nombAccion = "Apilar";
                comAccion = "Apilar";

              jMenuFunc1.setActionCommand(comAccion);
              jMenuFunc1.setText(nombAccion);
              jMenuFunc1.setEnabled(false);
              jMenuFunc1.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(ActionEvent e) {
                  jMenuFuncNuevo_actionPerformed(e);
                }
              });

                nombAccion = "Desapilar";
                comAccion = "Desapilar";

              jMenuFunc2.setActionCommand(comAccion);
              jMenuFunc2.setText(nombAccion);
              jMenuFunc2.setEnabled(false);
              jMenuFunc2.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(ActionEvent e) {
                  jMenuFuncQuitarElemento_actionPerformed(e);
                }
              });

                nombAccion = "Cima";
                comAccion = "Cima";

              jMenuFunc3.setActionCommand(comAccion);
              jMenuFunc3.setText(nombAccion);
              jMenuFunc3.setEnabled(false);
              jMenuFunc3.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(ActionEvent e) {
                  jMenuFunc3_actionPerformed(e);
                }
              });

               //resto de funciones de las pilas
                  .
                  .
                  .
                  .

        }
```

- **Clase VentanaSelección:**
  El método de inicialización de la ventana que se mostraba al usuario para que eligiese la ED o algoritmo con el que deseaba trabajar era común a los dos, obligando a distinguir casos para cada tipo.
  Por ejemplo, dicha clase contenía entre otros los siguientes métodos:
  - El método *comboED_itemStateChanged* se encargaba de discriminar la ED elegida por el usuario y en función de ella crear la interfaz gráfica con el usuario.

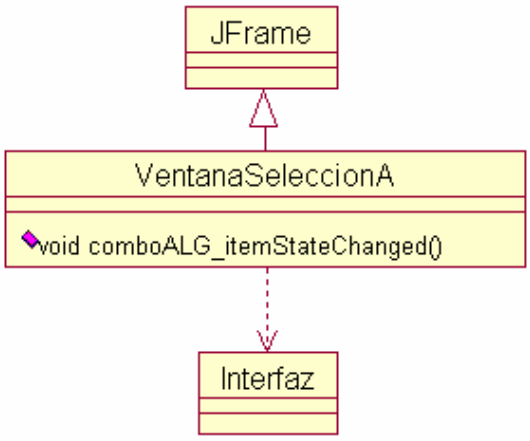
```
void comboED_itemStateChanged(ItemEvent e) {
   Interfaz frame = null;
   if (comboED.getSelectedItem().equals("Pila"))
     frame = new Interfaz("Pila", this);
   if (comboED.getSelectedItem().equals("Cola"))
     frame = new Interfaz("Cola", this);
   if (comboED.getSelectedItem().equals("Árbol Binario Búsqueda"))
     frame = new Interfaz("Arbol", this);
   if (comboED.getSelectedItem().equals("Árbol Equilibrado AVL"))
     frame = new Interfaz("AVL", this);
   if (comboED.getSelectedItem().equals("Cola de Prioridad"))
     frame = new Interfaz("Cola de Prioridad", this);
          .
          .
          .
   frame.setVisible(true);
   comboED.setSelectedItem(" ");
   this.setVisible(false);
 }
```

  Como se puede observar en el código, era necesario distinguir casos para cada EDs y pasarle la información al método *Interfaz* de cual es la ED seleccionada por el usuario.

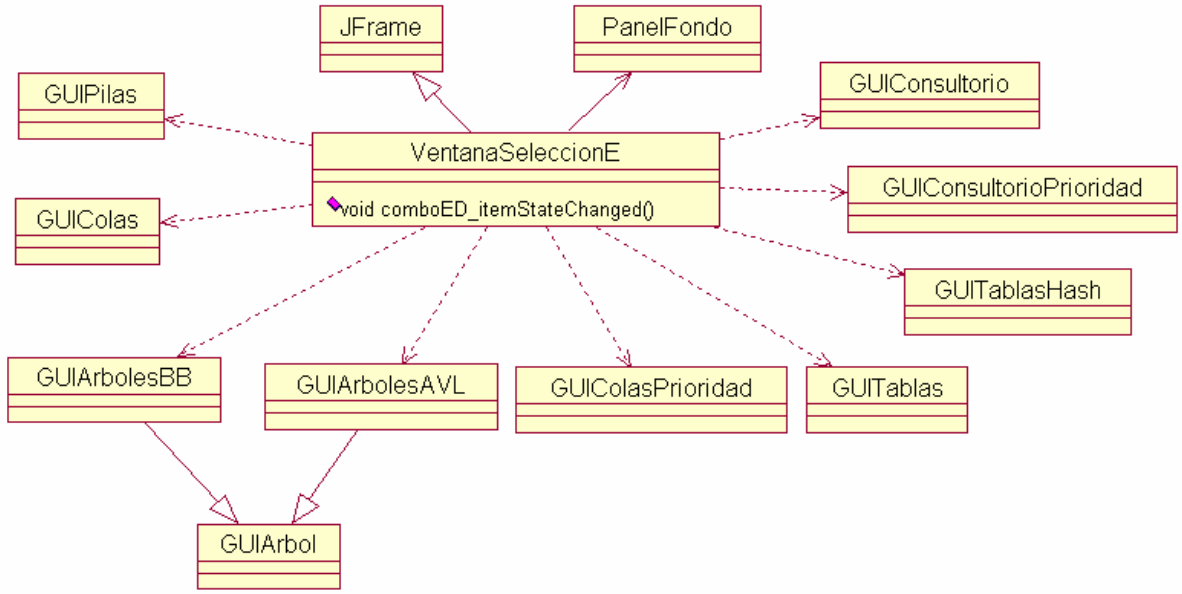
  - El método *comboALG_itemStateChanged* es simétrico al método anterior. Se encargaba de discriminar qué algoritmo había sido elegido por el usuario y en función de la elección, crear la interfaz gráfica con el usuario.

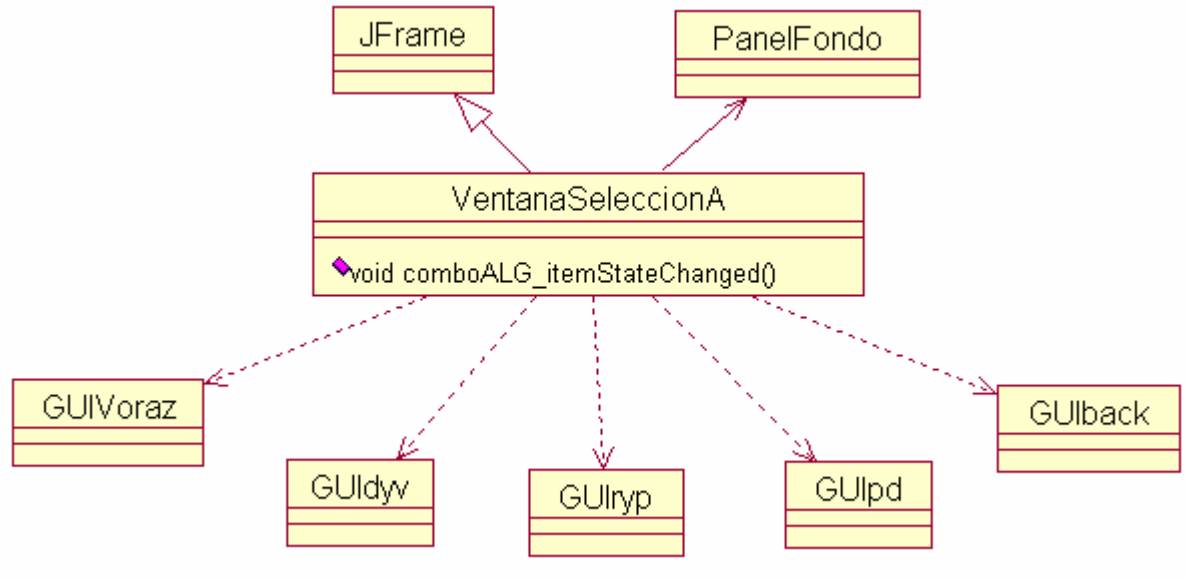
```
void comboALG_itemStateChanged(ItemEvent e) {
   Interfaz frame = null;
   if (comboALG.getSelectedItem().equals("Voraz"))
     frame = new Interfaz("Voraz", this);
   if (comboALG.getSelectedItem().equals("Programación Dinámica"))
     frame = new Interfaz("PD", this);
   if (comboALG.getSelectedItem().equals("Divide y vencerás"))
     frame = new Interfaz("DV", this);
   if (comboALG.getSelectedItem().equals("Ramificación y Poda"))
     frame = new Interfaz("RP", this);
   if (comboALG.getSelectedItem().equals("Vuelta atrás"))
     frame = new Interfaz("VA", this);
   frame.setVisible(true);
   comboALG.setSelectedItem(" ");
   this.setVisible(false);
 }
```

En resumen, la relación entre las clases VentanaSeleccion e Interfaz se muestra en el siguiente diagrama de clases:



Lo que se tiene después de modularizar son dos clases Ventana de Selección separadas, una para EDs (VentanaSelecciónE) y otra para algoritmos (VentanaSelecciónA). A continuación se muestra un diagrama de clases para explicar este hecho:


JFrame
PanelFondo
GUIPilas
GUIConsultorio
VentanaSeleccionE
void comboED_itemStateChanged()
GUIColas
GUIConsultorioPrioridad
GUITablasHash
GUIArbolesBB
GUIArbolesAVL
GUIColasPrioridad
GUITablas
GUIArbol


JFrame
PanelFondo
VentanaSeleccionA
void comboALG_itemStateChanged()
GUIVoraz
GUIback
GUIdyv
GUIryp
GUIpd

5.1.4. Algunas cifras:

A continuación se dan algunos datos sobre el tamaño medido en LDC para los métodos más importantes (y comunes a la mayoría de las clases) que intervienen en la construcción de la interfaz gráfica de la aplicación con el usuario, centrándonos sobre todo en los métodos de las EDs ya que las diferencias son más notables.

Método	**Antes de modularizar (LDC)**	**Clase**	**Después de modularizar (LDC)**
jbInit()	1949	GUIPilas	819
		GUIColas	778
		GUIColasPrioridad	811
		GUIArbolBB	1059
		GUIArbolAVL	1040
		GUITablas	972
		GUITablasHash	1005
jMenuFuncCrear_actionPerformed()	403	GUIPilas	218
		GUIColas	217
		GUIColasPrioridad	176
		GUIArbolBB	193
		GUIArbolAVL	150
		GUITablas	168
		GUITablasHash	163
nuevoElemento ()	303	GUIPilas	119
		GUIColas	113
		GUIColasPrioridad	127
		GUIArbolBB	138
		GUIArbolAVL	143

		GUITablas	152
		GUITablasHash	149
jMenuFuncQuitarElemento_actionPerformed()	196	GUIPilas	86
		GUIColas	94
		GUIColasPrioridad	65
		GUIArbolBB	32
		GUIArbolAVL	40
		GUITablas	41
		GUITablasHash	49

Aunque la cantidad total de LDC es mayor, tiene las siguientes ventajas:

- Mayor modularidad.
- Mayor facilidad de ampliación.
- Mayor reusabilidad.
- Mayor independencia entre clases.
- Respuesta lineal a cambios: un cambio pequeño en la especificación produce un cambio pequeño en el sistema.

5.2. Extensión de la herramienta

Durante el curso 2005/2006 esta herramienta ha sido ampliada con las siguientes estructuras:

Tabla Ordenada:

Implementada con árboles binarios de búsqueda. Consta de dos vistas, la de usuario y la de implementación. Permite las operaciones de crear, insertar un elemento, eliminar un elemento, consultar el valor de un elemento, consultar la existencia de un elemento, consultar si la tabla está vacía y recorrer la tabla en inorden. El comportamiento de estas operaciones se puede ver en el apartado de casos de uso.

Tabla Hash

Implementada con arrays de nodos. Consta de tres vistas, usuario, implementación abierta e implementación cerrada. Permite las operaciones de crear, insertar un elemento, eliminar un elemento, consultar el valor de un elemento, consultar la existencia de un elemento y consultar si la tabla está vacía. El comportamiento de estas operaciones se puede ver en el apartado de casos de uso.

Consultorio médico

Implementado mediante una tabla hash cerrada y colas circulares asociadas a cada clave de la tabla. Consta de dos vistas, usuario e implementación. Permite las operaciones de crear, dar de alta a un médico, pedir consulta, atender consulta, consultar el siguiente paciente de un médico y consultar si un médico tiene pacientes. El comportamiento de estas operaciones se puede ver en el apartado de casos de uso.

Consultorio médico de prioridad

Implementado mediante una tabla hash cerrada y montículos de mínimos asociados a cada clave de la tabla. Consta de dos vistas, usuario e implementación. Permite las operaciones de crear, dar de alta a un médico, pedir consulta, atender consulta, consultar el siguiente paciente de un médico y consultar si un médico tiene pacientes. El comportamiento de estas operaciones se puede ver en el apartado de casos de uso.

5.3. Cambios en las ventanas de presentación y selección

Los principales cambios han sido:

- La apariencia de las ventanas principal y de selección.
- En la ventana principal es donde se elige si se desea trabajar con una ED o con un algoritmo.
- Ventanas de selección separadas para las EDs y para los algoritmos.

La motivación para esta mejora no ha sido más que una cuestión de estética.

A continuación mostramos la apariencia de las ventanas mencionadas:

Ventana principal: el aspecto de la ventana que se muestra al usuario nada más comenzar la aplicación es el que se muestra a continuación:

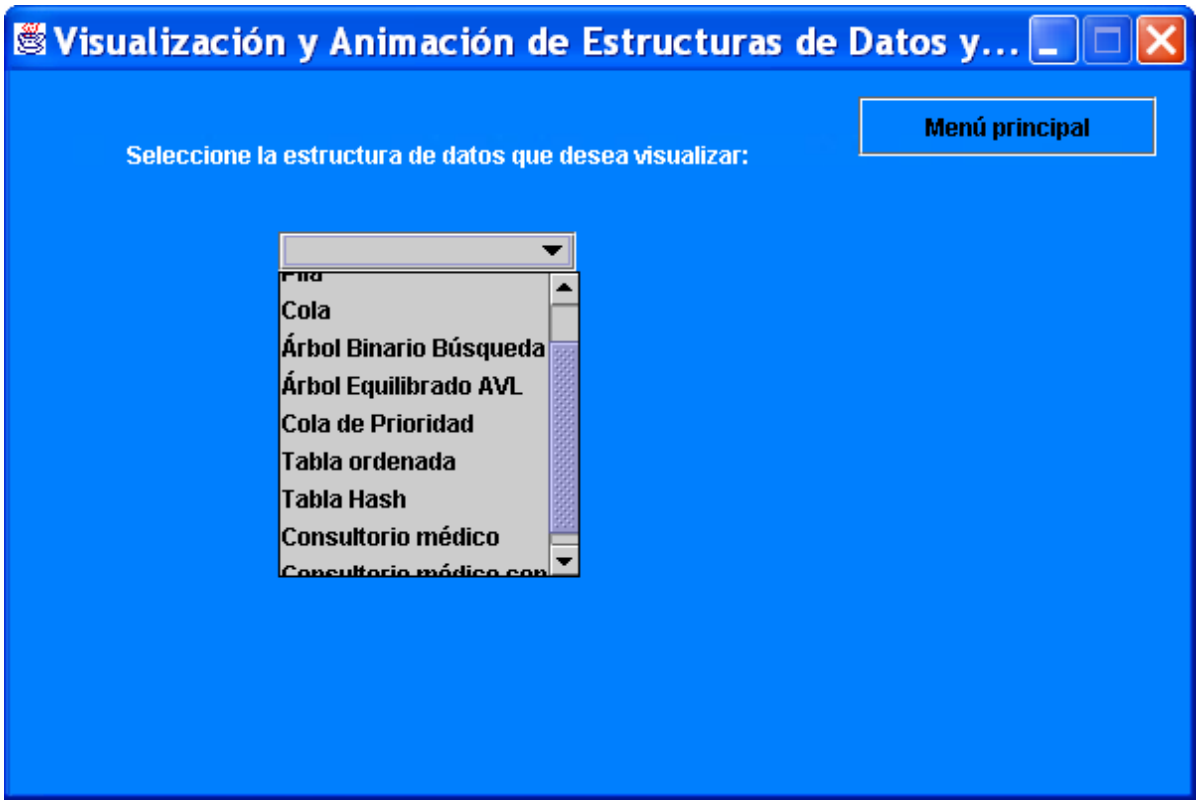
Botón "Estructuras de datos": da acceso a la ventana de selección de los distintos tipos de EDs de la aplicación.

Botón "Introducción": da la bienvenida a los usuarios de la aplicación y muestra un resumen de la aplicación y de su funcionalidad.

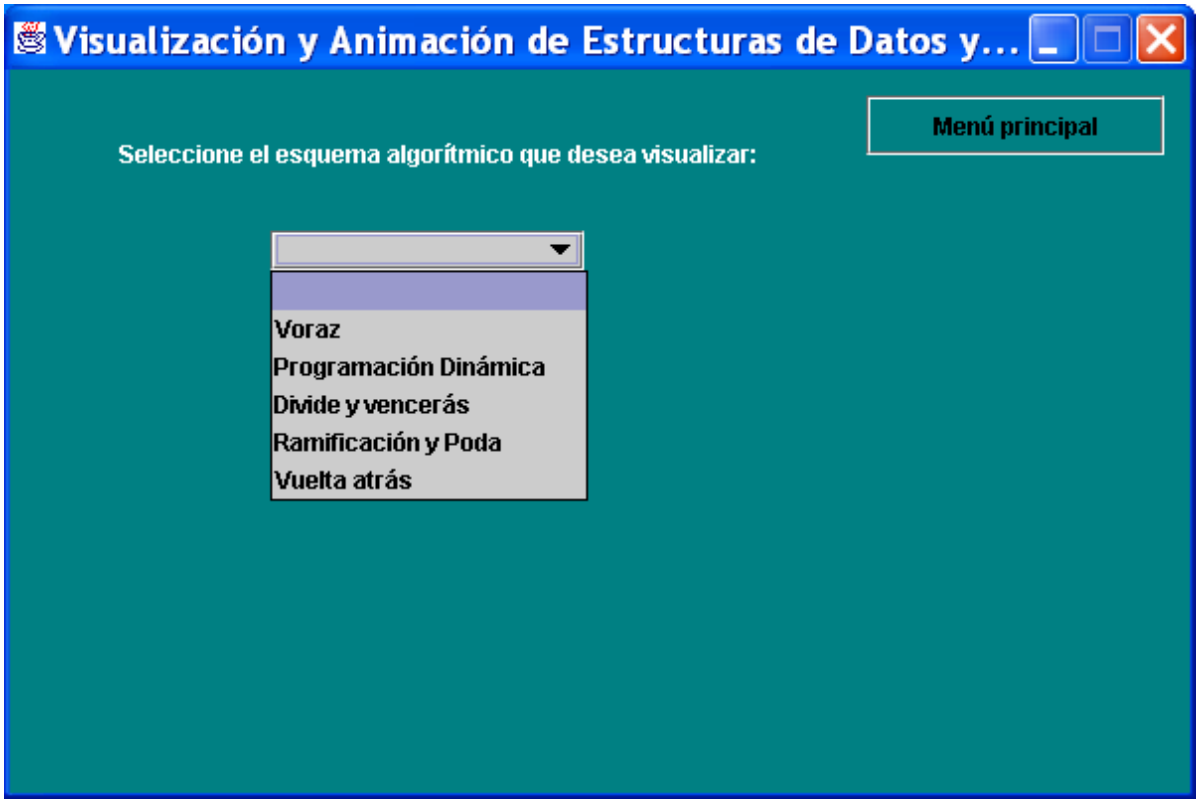
Botón "Métodos algorítmicos": da acceso a la ventana de selección de los distintos tipos de algoritmos de la aplicación.

Botón "Créditos": muestra los nombres de los desarrolladores de la herramienta.

Ventana de selección de las EDs:



Ventana de selección de los algoritmos:



CASOS DE USO

6.1 Comentario sobre los casos de uso 2004/2005

Nuestra aplicación es una extensión de otra elaborada en el curso 2004-05. Debido a este motivo, existen casos de uso diseñados en el curso anterior. La descripción de los mismos se encuentra en la memoria realizada y entregada en dicho año, por lo tanto, viendo innecesario su repetición, se ha decidido explicar a continuación sólo los casos de uso correspondientes a la versión de esta herramienta realizada en el presente curso 2005-06.

6.2 Casos de uso del año 2005/2006

6.2.1. Comienzo de la Aplicación

CASO DE USO #1	**Inicio de aplicación**	
Objetivo en contexto	Visualización de una pantalla de selección de las diversas opciones de la aplicación.	
Entradas	No hay.	
Precondiciones	El usuario arranca la aplicación.	
Salidas	Visualización de una pantalla de selección de las distintas opciones que tiene la aplicación.	
Postcondición si éxito	No hay.	
Postcondición si fallo	No hay.	
Actores	Usuario de la herramienta o usuario de desarrollo.	
Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se muestran las distintas opciones de la aplicación.

CASO DE USO #2	**Créditos**	
Objetivo en contexto	Mostrar los nombres de las personas que han realizado esta aplicación.	
Entradas	No hay.	
Precondiciones	El usuario ha seleccionado la opción de créditos.	
Salidas	Se muestran las personas que ha realizado esta aplicación.	
Postcondición si éxito	No hay.	
Postcondición si fallo	No hay.	
Actores	Usuario.	
Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se muestran las personas que han realizado la aplicación.

CASO DE USO #3	**Estructuras de datos**	
Objetivo en contexto	Seleccionar la estructura de datos con la que se quiere trabajar.	
Entradas	Selección de la estructura de datos.	
Precondiciones	El usuario ha seleccionado la opción de estructuras de datos.	
Salidas	Se muestra una lista con las distintas estructuras de datos disponibles en la aplicación.	
Postcondición si éxito	Se muestra la interfaz de la estructura de datos seleccionada.	
Postcondición si fallo	No hay.	
Actores	Usuario.	
Secuencia normal		
Paso	**Acción**	**Respuesta**

1.		Se muestra la interfaz de la estructura de datos seleccionada.

CASO DE USO #4	**Métodos algorítmicos**	
Objetivo en contexto	Seleccionar el método algorítmico con el que se quiere trabajar.	
Entradas	Selección del método algorítmico.	
Precondiciones	El usuario ha seleccionado la opción de métodos algorítmicos.	
Salidas	Se muestra una lista con los distintos métodos algorítmicos disponibles en la aplicación.	
Postcondición si éxito	Se muestra la interfaz del método algorítmico seleccionado.	
Postcondición si fallo	No hay.	
Actores	Usuario.	
Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se muestra la interfaz del método algorítmico.

CASO DE USO #5	**Introducción**	
Objetivo en contexto	Ver la introducción de la aplicación.	
Entradas	No hay.	
Precondiciones	El usuario ha seleccionado la opción de introducción.	
Salidas	Se muestra la introducción a la aplicación.	
Postcondición si éxito	Se muestra la introducción a la aplicación.	
Postcondición si fallo	No hay.	
Actores	Usuario.	
Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se muestra una introducción al proyecto.

6.2.2. Finalizar la aplicación

CASO DE USO #6	**Salir de la aplicación**	
Objetivo en contexto	Salir de la aplicación.	
Entradas	No hay.	
Precondiciones	El usuario se encuentra en una de las ventanas de la aplicación.	
Salidas	Se cierra la ventana de la aplicación donde se encontrase el usuario.	
Postcondición si éxito	Se sale de la aplicación.	
Postcondición si fallo	No hay.	
Actores	Usuario de la herramienta o usuario de desarrollo.	
Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se sale de la aplicación.

6.2.3. Configuración de la aplicación

CASO DE USO #7	Visualizar elementos de la interfaz	
Objetivo en contexto	Seleccionar si se desea tener visible o no en la interfaz una serie de elementos: descripción de la acción, estado actual, estado anterior y funciones.	
Entradas	Selección de visualización de elementos.	
Precondiciones	El usuario se encuentra en la ventana principal correspondiente a la ED o esquema algorítmico concreto.	
Salidas	Activación o desactivación de la visualización seleccionada por el usuario de los elementos de la interfaz.	
Postcondición si éxito	No hay.	
Postcondición si fallo	No hay.	
Actores	Usuario de la herramienta o usuario de desarrollo.	
Secuencia normal		
Paso	**Acción**	**Respuesta**
1.	Seleccionar los elementos que se deseen visualizar.	Se muestran los elementos que se desean visualizar.

6.2.4. Simulación

CASO DE USO #8	**Ver simulación**
Objetivo en contexto	Visualizar de forma animada el comportamiento de las EDs y los algoritmos ante unos ejemplos concretos dados, sin tener que introducir los datos desde la interfaz para observar dicho comportamiento.
Entradas	No hay.
Precondiciones	El usuario se encuentra en la pantalla principal de una ED o de un problema concreto que sigue la metodología de un esquema algorítmico.
Salidas	Animación del comportamiento de la ED o del algoritmo de acuerdo con el ejemplo de simulación elegido.
Postcondición si éxito	Se realizan las operaciones del fichero sobre la EDs o se ejecuta el algoritmo con los datos de entrada almacenados en la aplicación.
Postcondición si fallo	No hay.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se muestra una animación de la estructura de datos o método algorítmico en el que nos encontremos.

6.2.5. Documentación

CASO DE USO #9	**Mostrar especificación de la estructura de datos**
Objetivo en contexto	Consultar la especificación de la estructura de datos, la cual indica las operaciones que pueden realizarse sobre la estructura, de manera independiente de la implementación.
Entradas	Ninguna.
Precondiciones	El usuario se encuentra en la estructura de datos de la que se quiere obtener la especificación.
Salidas	Especificación de la estructura de datos.
Postcondición si éxito	No hay.
Postcondición si fallo	Ninguna.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	**Acción**	**Respuesta**
1.		Se muestra la documentación de la estructura de datos pedida.

CASO DE USO #10	**Mostrar código fuente de la estructura de datos**
Objetivo en contexto	Se mostrará el código con el que se ha implementado la estructura, dependiendo de la implementación escogida en la vista usuario de desarrollo.
Entradas	Ninguna.
Precondiciones	El usuario se encuentra en la ventana principal de la estructura de datos de la que se quiere obtener el código.
Salidas	Código fuente en Java de la estructura de datos, en función de la implementación escogida.
Postcondición si éxito	No hay.
Postcondición si fallo	Ninguna.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	**Acción**	**Respuesta**
1.		Se muestra el código fuente de la estructura de datos pedida.

CASO DE USO #11	**Mostrar coste de la estructura de datos**
Objetivo en contexto	Visualizar el coste de las operaciones de la estructura de datos.
Entradas	Ninguna.
Precondiciones	El usuario se encuentra en la estructura de datos de la que se quiere obtener el coste.
Salidas	Visualización del coste asintótico temporal y espacial de cada una de las operaciones de la estructura de datos escogida en las diferentes implementaciones.
Postcondición si éxito	No hay
Postcondición si fallo	Ninguna.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	**Acción**	**Respuesta**
1.		Se muestra el coste de la estructura de datos pedida.

CASO DE USO #12	**Mostrar ayuda adicional de la estructura de datos**
Objetivo en contexto	Se dispondrá de una página de ayuda que contenga información general sobre la estructura de datos (definición de la estructura de datos, características, operaciones permitidas, etc.).
Entradas	Ninguna.
Precondiciones	El usuario se encuentra en la estructura de datos de la que se quiere obtener la ayuda adicional.
Salidas	Ayuda con información general sobre la estructura de datos.
Postcondición si éxito	No hay
Postcondición si fallo	Ninguna.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	**Acción**	**Respuesta**
1.		Se muestra una ayuda acerca de la estructura de datos pedida.

6.2.6. Visualización y Animación de Tabla Ordenada

CASO DE USO #13	**Crear tabla ordenada**
Objetivo en contexto	Crear una nueva tabla ordenada del tipo de la vista e implementación escogida, para poder realizar operaciones sobre la tabla ordenada.
Entradas	Tipo de la clave y el valor de la tabla ordenada que se desea crear.
Precondiciones	El usuario se encuentra en la parte de la aplicación de la estructura de datos tabla ordenada.
Salidas	Si no se cumple alguna de las condiciones de la precondición, se muestra un mensaje de error, si no, se dibuja la tabla ordenada. En el caso de que la tabla ordenada ya se hubiera creado con anterioridad, se mostrará un mensaje para confirmar si se quiere comenzar de nuevo.
Postcondición si éxito	Se creará una nueva tabla ordenada sobre la que se pueden realizar todas las operaciones.
Postcondición si fallo	Se mostrará un mensaje informando del motivo del fallo.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	Acción	Respuesta
1.	Se introducen el tipo de clave y valor de la tabla.	Se crea una tabla ordenada con la clave y valor indicados. Si la tabla ya existe se crea una nueva.

CASO DE USO #14	**Insertar un elemento en la tabla ordenada**
Objetivo en contexto	Añadir un nuevo elemento a la tabla ordenada creada con anterioridad.
Entradas	Clave y valor a añadir.
Precondiciones	La tabla ordenada ha sido creada y las entradas son del tipo especificado al crearla.
Salidas	Se visualiza la inserción del nuevo elemento en el lugar correspondiente y se reordenan los restantes elementos de la tabla ordenada.
Postcondición si	Se inserta el nuevo elemento en el lugar correspondiente y se reordenan

éxito	los elementos de la tabla ordenada.
Postcondición si fallo	La tabla ordenada permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción, es decir, el elemento no se inserta.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	**Acción**	**Respuesta**
1.		Se comprueba que la tabla ordenada se ha creado con anterioridad.
1a.	La tabla ha sido creada.	Comprobar que las entradas introducidas son del tipo especificado al crear la tabla ordenada, ir a 1.c.
1b.	La tabla no ha sido creada.	Ir a S1.
1.c.	Las entradas introducidas son del tipo especificado al crear la tabla y la clave no se encuentra en la tabla.	Se añade el elemento a la tabla ordenada, ir a 2.
1.d.	Las entradas introducidas no son del tipo especificado al crear la tabla.	Ir a S2.
1.e	La clave introducida ya se encuentra en la tabla.	Ir a S3.
2.	El usuario se encuentra en la vista de usuario de la herramienta.	Se inserta la clave y el valor en la tabla ordenada.

Secuencia alternativa

Paso	**Acción**	**Respuesta**
S1.	Tabla ordenada no creada.	Se muestra un mensaje de error que indique que la tabla ordenada no ha sido creada.
S2.	Las entradas no son del tipo especificado al crear la tabla.	Si muestra un mensaje de error informando sobre la incompatibilidad de tipos y no se añade el elemento introducido.
S3.	La clave introducida ya se encuentra en la tabla.	Si muestra un mensaje de error informando sobre la existencia de esa clave en la tabla ordenada y no se añade el elemento introducido.

CASO DE USO #15	**Eliminar un elemento de la tabla ordenada**
Objetivo en contexto	Eliminar un elemento de la tabla ordenada creada con anterioridad.
Entradas	Elemento a eliminar.
Precondiciones	La tabla ordenada ha sido creada y existe algún elemento en la misma.
Salidas	Se elimina el elemento de la tabla ordenada y se reordenan los restantes elementos.
Postcondición si éxito	Se elimina el elemento de la tabla ordenada y se reordenan los restantes elementos.
Postcondición si fallo	La tabla ordenada permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción, es decir, el elemento no se elimina.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	**Acción**	**Respuesta**
1.		Se comprueba que la tabla ordenada se ha creado con anterioridad.
1a.	La tabla ordenada ha sido creada.	Comprobar que la tabla ordenada no esté vacía, ir a 1c.
1b.	La tabla ordenada no ha sido creada.	Ir a S1.
1c.	La tabla ordenada no está vacía.	Se comprueba que el elemento introducido por el usuario pertenece a tabla ordenada, ir a 1d.
1d.	El elemento introducido por el usuario pertenece a la tabla ordenada.	Se elimina el elemento de la tabla ordenada interna. Ir a 2.
1e.	El elemento introducido por el usuario no pertenece a la tabla ordenada.	Ir a S2.
2.	El usuario se encuentra en la vista de usuario de la herramienta.	Se elimina el elemento en cuestión.

Secuencia alternativa

Paso	**Acción**	**Respuesta**
S1.	Tabla ordenada no creada.	Se muestra un mensaje de error que indique que la tabla ordenada no ha sido creada.

S2.	El elemento a eliminar no existe.	Se mostrará un mensaje informando sobre la imposibilidad de eliminar el elemento debido a la inexistencia del mismo en la tabla ordenada.

CASO DE USO #16	**Consultar el valor de un elemento en la tabla ordenada**	
Objetivo en contexto	Consultar el valor un elemento en la tabla ordenada.	
Entradas	Elemento a consultar.	
Precondiciones	La tabla ordenada ha sido creada y existe algún elemento en la misma.	
Salidas	Se muestra el valor de asociado a la clave consultada.	
Postcondición si éxito	Se visualizará el valor asociado a la clave consultada.	
Postcondición si fallo	La tabla ordenada permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción. Se mostrará un mensaje de error.	
Actores	Usuario de la herramienta o usuario de desarrollo.	
Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se comprueba que la tabla ordenada ha sido creada.
1a.	La tabla ordenada ha sido creada y existe el elemento a consultar.	Se muestra el valor asociado al elemento introducido.
1b.	La tabla ordenada no ha sido creada.	Ir a S1.
1c.	El elemento introducido no existe en la tabla ordenada.	Ir a S2.
Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Tabla ordenada no creada.	Se muestra un mensaje de error que indique que la tabla ordenada no ha sido creada.
S2.	El elemento a consultar no existe.	Se mostrará un mensaje informando sobre la imposibilidad de consultar el elemento debido a la inexistencia del mismo en la tabla ordenada.

CASO DE USO #17	**Consultar la existencia de un elemento en la tabla ordenada**
Objetivo en contexto	Consultar la existencia de un elemento en la tabla ordenada.
Entradas	Elemento a consultar.
Precondiciones	La tabla ordenada ha sido creada y existe algún elemento en la misma.
Salidas	Se muestra so existe o no el elemento en la tabla ordenada.
Postcondición si éxito	No hay.
Postcondición si fallo	La tabla ordenada permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción. Se mostrará un mensaje de error.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se comprueba que la tabla ordenada ha sido creada.
1a.	La tabla ordenada ha sido creada y existe el elemento a consultar.	Se muestra la existencia del elemento en la tabla ordenada.
1b.	La tabla ordenada no ha sido creada.	Ir a S1.
1c.	La tabla ordenada ha sido creada y no existe el elemento a consultar.	Se muestra la no existencia del elemento en la tabla ordenada.
Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Tabla ordenada no ha sido creada.	Se muestra un mensaje de error que indique que la tabla ordenada no ha sido creada.

CASO DE USO #18	**Recorrer una tabla ordenada**
Objetivo en contexto	Recorrer en inorden una tabla ordenada.
Entradas	No hay.
Precondiciones	La tabla ordenada ha sido creada.
Salidas	La tabla ordenada permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción. Se muestra el recorrido.
Postcondición si éxito	No hay.
Postcondición si fallo	La tabla ordenada permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción. Se mostrará un mensaje de error.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	**Acción**	**Respuesta**
1.		Se comprueba la tabla ordenada ha sido creada.
1a.	La tabla ordenada ha sido creada.	Se comprueba que la tabla ordenada no sea vacía, ir a 1c.
1b.	La tabla ordenada no ha sido creada.	Ir a S1.
1c.	La tabla ordenada no es vacía.	Se recorre la tabla, ir a 2.
1d.	La tabla ordenada es vacía.	Ir a S2.
2.		Se recorre la tabla de forma ordenada.

Secuencia alternativa

Paso	**Acción**	**Respuesta**
S1.	Tabla ordenada no creada.	Se muestra un mensaje de error que indique que la tabla ordenada no ha sido creada.
S2.	Tabla ordenada vacía.	Se muestra un mensaje informando de que no se puede hacer un recorrido de una tabla ordenada vacía.

CASO DE USO #19	**Consultar si la tabla ordenada está vacía**
Objetivo en contexto	Consultar si la tabla ordenada está vacía.
Entradas	No hay.
Precondiciones	La tabla ordenada ha sido creada.
Salidas	La tabla ordenada permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción. Se muestra si la tabla ordenada está vacía o no.
Postcondición si éxito	No hay.
Postcondición si fallo	La tabla ordenada permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción. Se mostrará un mensaje de error.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	**Acción**	**Respuesta**
1.		Comprobar que la tabla ordenada ha sido creada.
1a.	La tabla ordenada ha sido creada.	Se muestra si la tabla ordenada está vacía o no.
1b.	La tabla ordenada no ha sido creada.	Ir a S2.

Secuencia alternativa

Paso	**Acción**	**Respuesta**
S1.	Tabla ordenada no creada.	Se muestra un mensaje de error que indique que la tabla ordenada no ha sido creada.

6.2.7. Visualización y Animación de Tabla Hash

CASO DE USO #20	**Crear tabla hash**
Objetivo en contexto	Crear una nueva tabla hash del tipo de la vista e implementación escogida, para poder realizar operaciones sobre la tabla hash.
Entradas	Tipo de la clave y el valor de la tabla hash que se desea crear.
Precondiciones	El usuario se encuentra en la parte de la aplicación de la estructura de datos tabla hash.
Salidas	Si no se cumple alguna de las condiciones de la precondición, se muestra un mensaje de error, si no, se crea la tabla hash. En el caso de que la tabla hash ya se hubiera creado con anterioridad, se mostrará un mensaje para confirmar si se quiere comenzar de nuevo.
Postcondición si éxito	Se crea una nueva tabla hash sobre la que se pueden realizar todas las operaciones.
Postcondición si fallo	Se muestra un mensaje informando del motivo del fallo.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	**Acción**	**Respuesta**
1.	Se introducen el tipo de clave y valor de la tabla.	Se crea una tabla hash con la clave y valor indicados. Si la tabla ya existe se crea una nueva.

CASO DE USO #21	**Insertar un elemento en la tabla hash**
Objetivo en contexto	Añadir un nuevo elemento a la tabla hash creada con anterioridad.
Entradas	Clave y valor a añadir.
Precondiciones	La tabla hash ha sido creada y las entradas son del tipo especificado al crearla.
Salidas	Se visualiza la inserción del nuevo elemento en el lugar correspondiente.
Postcondición si éxito	Se inserta el nuevo elemento en el lugar correspondiente.
Postcondición si fallo	La tabla hash permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción, es decir, el elemento no se inserta.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se comprueba que la tabla hash se ha creado con anterioridad.
1a.	La tabla ha sido creada.	Comprobar que las entradas introducidas son del tipo especificado al crear la tabla hash, ir a 1.c.
1b.	La tabla no ha sido creada.	Ir a S1.
1.c	Las entradas introducidas son del tipo especificado al crear la tabla.	Se añade el elemento a la tabla hash, ir a 2.
1.d.	Las entradas introducidas no son del tipo especificado al crear la tabla.	Ir a S2.
2.		Se inserta la clave y el valor en la tabla hash.

Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Tabla hash no creada.	Se muestra un mensaje de error que indique que la tabla hash no ha sido creada.
S2.	Las entradas no son del tipo especificado al crear la tabla.	Si muestra un mensaje de error informando sobre la incompatibilidad de tipos y no se añade el elemento introducido.

CASO DE USO #22	**Eliminar un elemento de la tabla hash**
Objetivo en contexto	Eliminar un elemento de la tabla hash creada con anterioridad
Entradas	Clave del elemento a eliminar
Precondiciones	La tabla hash ha sido creada y existe algún elemento en la misma.
Salidas	Se elimina el elemento de la tabla.
Postcondición si éxito	Se elimina el elemento de la tabla hash.
Postcondición si fallo	La tabla hash permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción, es decir, el elemento no se elimina
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se comprueba que la tabla hash se ha creado con anterioridad.
1a.	La tabla hash ha sido creada	Comprobar que la tabla hash no esté vacía, ir a 1c.
1b.	La tabla hash no se ha sido creada.	Ir a S1
1c.	La tabla hash no está vacía	Se comprueba que el elemento introducido por el usuario pertenece a tabla hash, ir a 1d
1d.	El elemento introducido por el usuario pertenece a la tabla hash	Se elimina el elemento de la tabla hash interna. Ir a 2.
1e.	El elemento introducido por el usuario no pertenece a la tabla hash	Ir a S2.
2.		Se elimina el elemento en cuestión de la tabla hash.
Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Tabla hash no creada	Se muestra un mensaje de error que indique que la tabla hash no ha sido creada.
S2.	El elemento a eliminar no existe	Se mostrará un mensaje informando sobre la imposibilidad de eliminar el elemento debido a la inexistencia del mismo en la tabla hash.

CASO DE USO #23	**Consultar el valor de un elemento en la tabla hash**
Objetivo en contexto	Consultar el valor un elemento en la tabla hash.
Entradas	Elemento a consultar.
Precondiciones	La tabla hash ha sido creada y existe algún elemento en la misma.
Salidas	Se muestra el valor de asociado a la clave consultada.
Postcondición si éxito	No hay.
Postcondición si fallo	La tabla hash permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción. Se mostrará un mensaje de error.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se comprueba que la tabla hash ha sido creada.
1a.	La tabla hash ha sido creada y existe el elemento a consultar.	Se muestra el valor asociado al elemento introducido.
1b.	La tabla hash no ha sido creada.	Ir a S1.
1c.	El elemento introducido no existe en la tabla hash.	Ir a S2.

Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Tabla hash no creada.	Se muestra un mensaje de error que indique que la tabla hash no ha sido creada.
S2.	El elemento a consultar no existe.	Se mostrará un mensaje informando sobre la imposibilidad de consultar el elemento debido a la inexistencia del mismo en la tabla hash.

CASO DE USO #24	**Consultar la existencia de un elemento en la tabla hash**
Objetivo en contexto	Consultar la existencia de un elemento en la tabla hash.
Entradas	Elemento a consultar.
Precondiciones	La tabla hash ha sido creada y existe algún elemento en la misma.
Salidas	Se muestra so existe o no el elemento en la tabla hash.
Postcondición si éxito	No hay.
Postcondición si fallo	La tabla hash permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción. Se mostrará un mensaje de error.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se comprueba que la tabla hash ha sido creada.
1a.	La tabla hash ha sido creada y existe el elemento a consultar.	Se muestra la existencia del elemento en la tabla hash.
1b.	La tabla hash no ha sido creada.	Ir a S1.
1c.	La tabla hash ha sido creada y no existe el elemento a consultar.	Se muestra la no existencia del elemento en la tabla hash.

Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Tabla hash no creada	Se muestra un mensaje de error que indique que la tabla hash no ha sido creada.

CASO DE USO #25	**Consultar si la tabla hash está vacía**
Objetivo en contexto	Consultar si la tabla hash está vacía.
Entradas	No hay
Precondiciones	La tabla hash ha sido creada
Salidas	La tabla hash permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción. Se muestra si la tabla hash está vacía o no.
Postcondición si éxito	No hay
Postcondición si fallo	La tabla hash permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción. Se mostrará un mensaje de error.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Comprobar que la tabla hash ha sido creada.
1a.	La tabla hash ha sido creada	Se muestra si la tabla hash está vacía o no.
1b.	La tabla hash no ha sido creada	Ir a S2

Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Tabla hash no creada	Se muestra un mensaje de error que indique que la tabla hash no ha sido creada.

6.2.8. Visualización y Animación del TAD

Por su similitud se ha decidido realizar los casos de uso del consultorio médico y del consultorio médico de prioridad en un solo, en los sucesivos casos de uso nos referiremos a estos tipos abstractos de datos como consultorio.

CASO DE USO #26	**Crear consultorio**	
Objetivo en contexto	Crear una nueva tabla consultorio del tipo de la vista e implementación escogida, para poder realizar operaciones sobre la tabla consultorio.	
Entradas	Tipo de la clave y el valor del consultorio que se desea crear.	
Precondiciones	El usuario se encuentra en la parte de la aplicación de la estructura de datos consultorio.	
Salidas	Si no se cumple alguna de las condiciones de la precondición, se muestra un mensaje de error, si no, se crea el consultorio. En el caso de que la tabla hash ya se hubiera creado con anterioridad, se mostrará un mensaje para confirmar si se quiere comenzar de nuevo.	
Postcondición si éxito	Se creará una nueva tabla hash sobre la que se pueden realizar todas las operaciones.	
Postcondición si fallo	Se mostrará un mensaje informando del motivo del fallo.	
Actores	Usuario de la herramienta o usuario de desarrollo.	
Secuencia normal		
Paso	**Acción**	**Respuesta**
1.	Se introducen el tipo de clave y valor del consultorio.	Se crea un consultorio con la clave y valor indicados. Si el consultorio ya existe se crea uno nuevo.

CASO DE USO #27	**Insertar un médico en el consultorio**
Objetivo en contexto	Añadir un nuevo médico a un consultorio creado con anterioridad.
Entradas	Clave del médico a añadir.
Precondiciones	El consultorio ha sido creado y la entrada es del tipo especificado al crearlo.
Salidas	Se muestra la inserción del médico en el consultorio.
Postcondición si éxito	Se inserta el nuevo médico.
Postcondición si fallo	El consultorio permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción, es decir, el elemento no se inserta.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se comprueba que el consultorio se ha creado con anterioridad.
1a.	El consultorio ha sido creado.	Comprobar que la entrada introducida es del tipo especificado al crear el consultorio, ir a 1.c
1b.	El consultorio no ha sido creado.	Ir a S1
1.c	La entrada introducida es del tipo especificado al crear el consultorio.	Comprobar el número de médicos existentes. Ir a 2.
1.d.	La entrada introducida no es del tipo especificado al crear el consultorio.	Ir a S2.
2.a.	Número de médicos existentes no es el máximo.	Se inserta el médico en el consultorio.
2.b.	Número de médicos existentes es el máximo.	Ir a S3.
Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Consultorio no creado.	Se muestra un mensaje de error que indique que el consultorio no ha sido creado.
S2.	La entrada no es del tipo especificado al crear el consultorio.	Si muestra un mensaje de error informando sobre la incompatibilidad de tipos y no se añade el elemento introducido.

S3.	Número de médicos existentes es el máximo.	Se muestra un mensaje indicando que no se pueden introducir más médicos debido a que se ha alcanzado el máximo.

CASO DE USO #28	**Pedir consulta en un consultorio**	
Objetivo en contexto	Añadir un nuevo paciente a un médico en un consultorio.	
Entradas	Clave del médico y valor del paciente.	
Precondiciones	El consultorio ha sido creado y las entrada son del tipo especificado al crearlo.	
Salidas	Se muestra la inserción del un paciente en un consultorio.	
Postcondición si éxito	Se inserta el nuevo paciente.	
Postcondición si fallo	El consultorio permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción, es decir, el elemento no se inserta.	
Actores	Usuario de la herramienta o usuario de desarrollo.	
Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se comprueba que el consultorio se ha creado con anterioridad.
1a.	El consultorio ha sido creado	Comprobar que las entradas introducidas son del tipo especificado al crear el consultorio, ir a 1.c.
1b.	El consultorio no ha sido creado.	Ir a S1.
1.c	Las entradas introducidas son del tipo especificado al crear el consultorio.	Se comprueba la existencia del médico, ir a 2.
1.d.	Las entradas introducidas no son del tipo especificado al el consultorio.	Ir a S2.
2.a.	El médico existe.	Se inserta el paciente con su médico.
2.b.	El médico no existe.	Ir a S3.
Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Consultorio no creado.	Se muestra un mensaje de error que indique que el consultorio no ha sido creado.
S2.	Las entradas no son del tipo especificado al crear el consultorio.	Si muestra un mensaje de error informando sobre la incompatibilidad de tipos y no se añade el elemento introducido.

S3.	No existe el médico.	Se muestra un mensaje indicando la imposibilidad de insertar un paciente con un médico no existente.

CASO DE USO #29	**Atender paciente en un consultorio**
Objetivo en contexto	Atender un paciente por un médico.
Entradas	Médico que atiende.
Precondiciones	El consultorio ha sido creado y existe algún elemento en el mismo.
Salidas	Se elimina el elemento del consultorio.
Postcondición si éxito	Se elimina el elemento del consultorio.
Postcondición si fallo	El consultorio permanece en el mismo estado en que se hallaba cuando se seleccionó esta opción, es decir, el elemento no se elimina
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se comprueba que el consultorio se ha creado con anterioridad.
1a.	El consultorio ha sido creado.	Comprobar que consultorio no esté vacío, ir a 1c.
1b.	El consultorio no ha sido creado.	Ir a S1.
1c.	El consultorio no está vacío.	Se comprueba las entradas introducidas son del tipo especificado al crear el consultorio, ir a 1d.
1d.	Las entradas introducidas son del tipo especificado al crear el consultorio.	Se comprueba la existencia del médico, ir a 2.
1e.	Las entradas introducidas no son del tipo especificado al el consultorio.	Ir a S2.
2a.	El médico pertenece al consultorio.	Se elimina el elemento del consultorio.
2b.	El médico no pertenece al consultorio.	Ir a S3.
Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Consultorio no creado.	Se muestra un mensaje de error que indique que el consultorio no ha sido creado.

S2.	Las entradas no son del tipo especificado al crear el consultorio.	Si muestra un mensaje de error informando sobre la incompatibilidad de tipos y no se añade el elemento introducido.
S3.	El elemento a eliminar no existe.	Se mostrará un mensaje informando sobre la imposibilidad de eliminar el elemento debido a la inexistencia del mismo.

CASO DE USO #30	**Siguiente paciente de un médico en un consultorio**
Objetivo en contexto	Consultar el siguiente paciente de un médico.
Entradas	Médico a consultar.
Precondiciones	El consultorio ha sido creado.
Salidas	Se muestra el siguiente paciente del médico introducido.
Postcondición si éxito	No hay.
Postcondición si fallo	No hay.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal

Paso	**Acción**	**Respuesta**
1.		Se comprueba que el consultorio se ha creado con anterioridad.
1a.	El consultorio ha sido creado.	Comprobar que consultorio no esté vacío, ir a 1c.
1b.	El consultorio no ha sido creado.	Ir a S1.
1c.	El consultorio no está vacío.	Se comprueba las entradas introducidas son del tipo especificado al crear el consultorio, ir a 1d.
1d.	Las entradas introducidas son del tipo especificado al crear el consultorio.	Se comprueba la existencia del médico, ir a 2.
1e.	Las entradas introducidas no son del tipo especificado al el consultorio.	Ir a S2.
2a.	El médico pertenece al consultorio.	Se muestra el siguiente paciente del médico introducido.
2b.	El médico no pertenece al consultorio.	Ir a S3.

Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Consultorio no creado.	Se muestra un mensaje de error que indique que el consultorio no ha sido creado.
S2.	Las entradas no son del tipo especificado al crear el consultorio.	Si muestra un mensaje de error informando sobre la incompatibilidad de tipos y no se añade el elemento introducido.
S3.	El elemento a consultar no existe.	Se mostrará un mensaje informando sobre la imposibilidad de consultar el elemento debido a la inexistencia del mismo.

CASO DE USO #31	**Tiene pacientes un médico en un consultorio**
Objetivo en contexto	Consultar si tiene pacientes un médico.
Entradas	Médico a consultar.
Precondiciones	El consultorio ha sido creado.
Salidas	Se muestra si tiene pacientes el médico introducido.
Postcondición si éxito	No hay.
Postcondición si fallo	No hay.
Actores	Usuario de la herramienta o usuario de desarrollo.

Secuencia normal		
Paso	**Acción**	**Respuesta**
1.		Se comprueba que el consultorio se ha creado con anterioridad.
1a.	El consultorio ha sido creado.	Comprobar que consultorio no esté vacío, ir a 1c.
1b.	El consultorio no ha sido creado.	Ir a S1.
1c.	El consultorio no está vacío.	Se comprueba las entradas introducidas son del tipo especificado al crear el consultorio, ir a 1d.
1d.	Las entradas introducidas son del tipo especificado al crear el consultorio.	Se comprueba la existencia del médico, ir a 2.

1e.	Las entradas introducidas no son del tipo especificado al el consultorio.	Ir a S2.
2a.	El médico pertenece al consultorio.	Se muestra si el médico introducido tiene pacientes o no.
2b.	El médico no pertenece al consultorio.	Ir a S3.
Secuencia alternativa		
Paso	**Acción**	**Respuesta**
S1.	Consultorio no creado.	Se muestra un mensaje de error que indique que el consultorio no ha sido creado.
S2.	Las entradas no son del tipo especificado al crear el consultorio.	Si muestra un mensaje de error informando sobre la incompatibilidad de tipos y no se añade el elemento introducido.
S3.	El elemento a consultar no existe.	Se mostrará un mensaje informando sobre la imposibilidad de consultar el elemento debido a la inexistencia del mismo.

7.1. Diagramas de casos de uso

TABLA ORDENADA

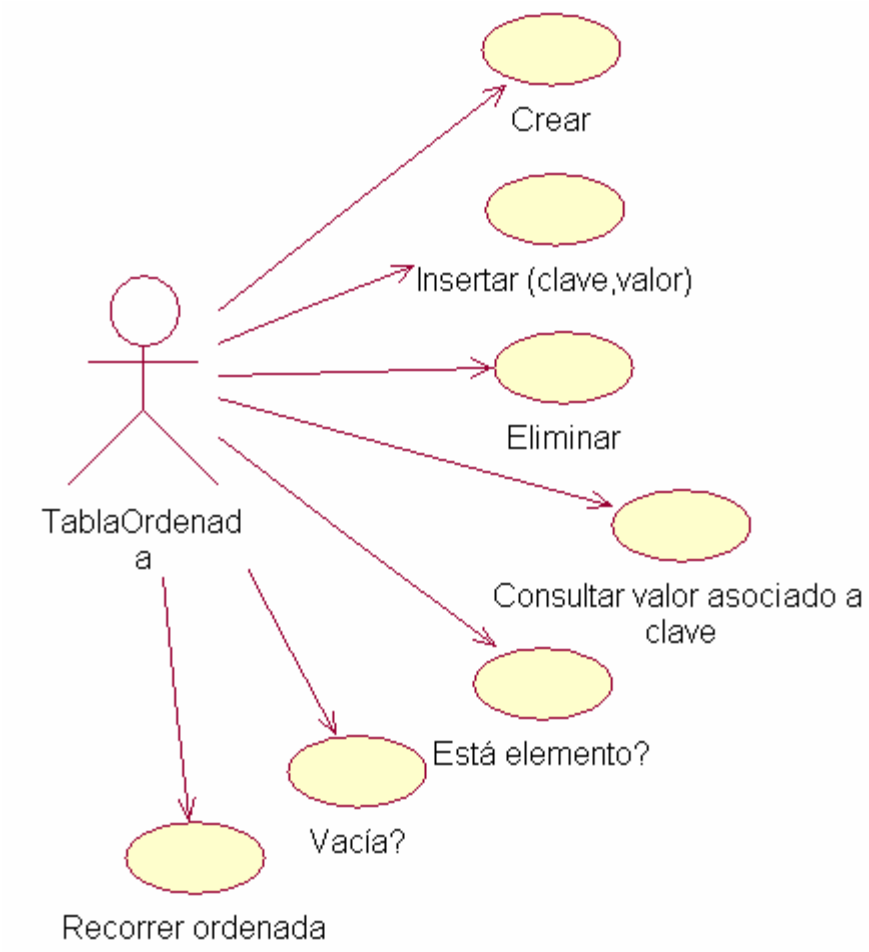
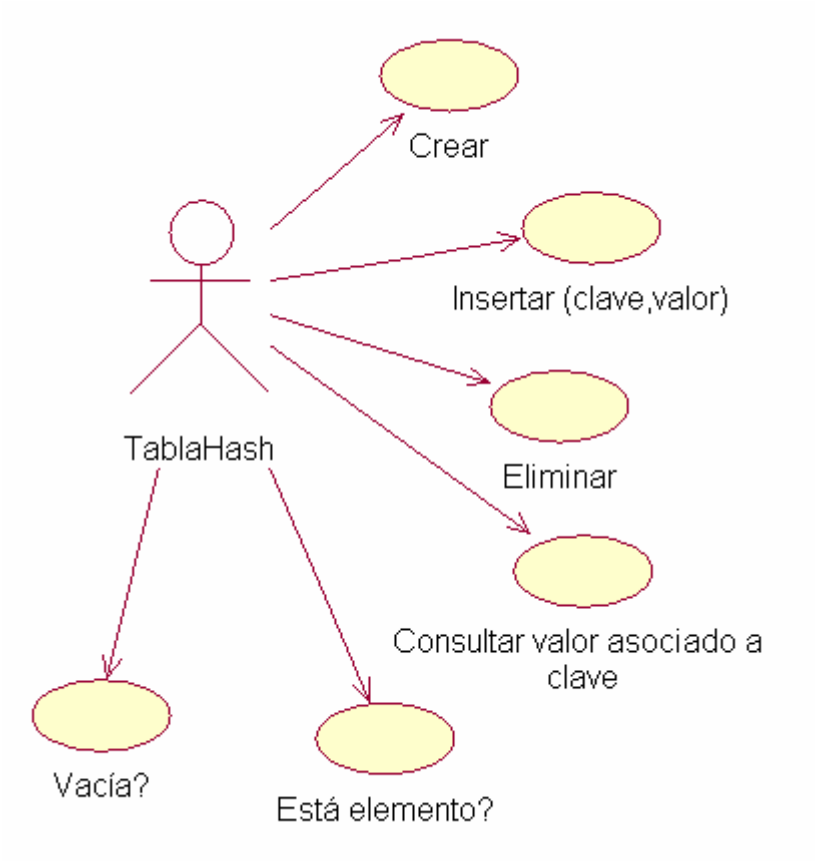
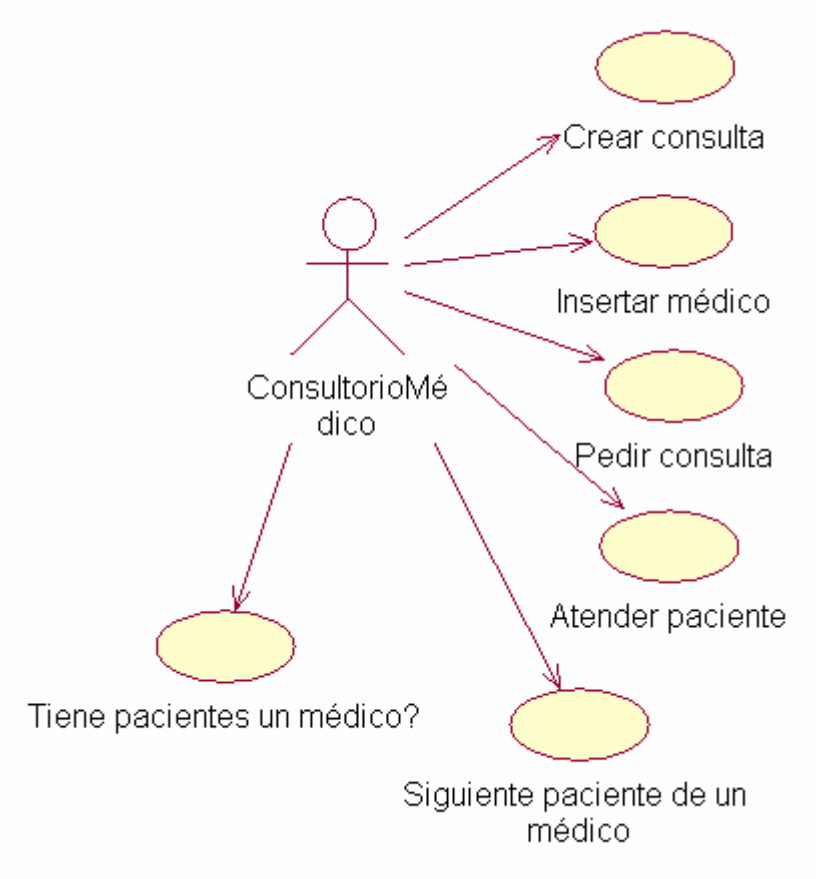


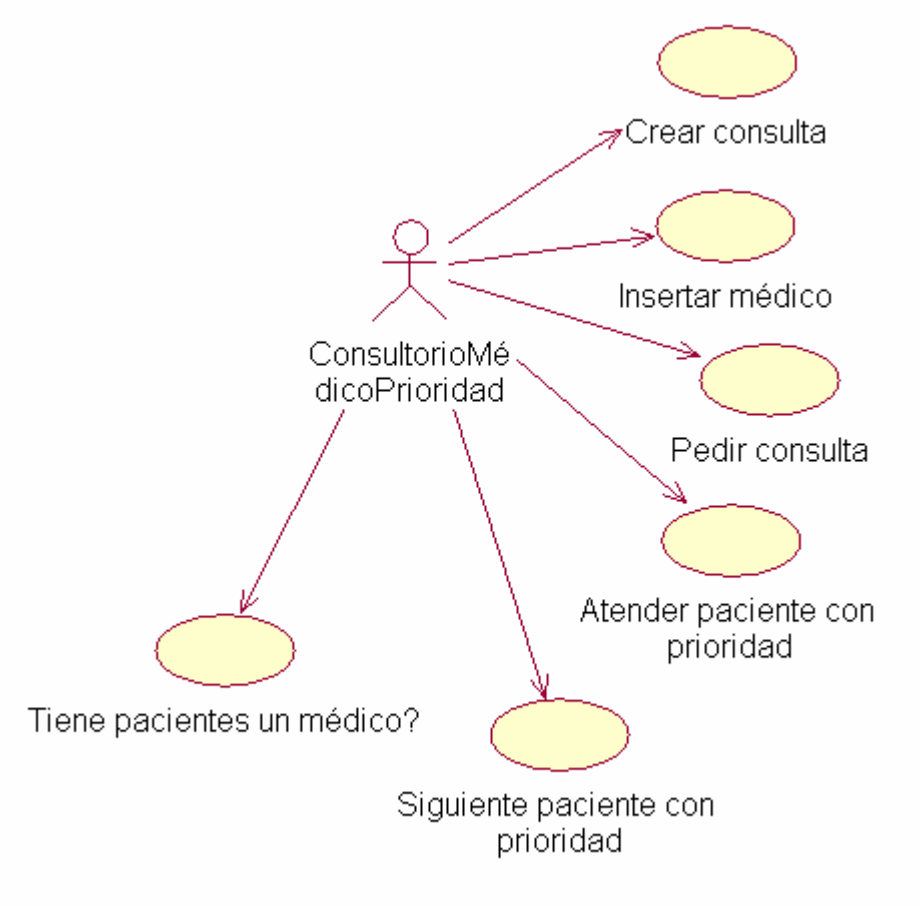
TABLA HASH


Crear
Insertar (clave,valor)
Eliminar
TablaHash
Consultar valor asociado a clave
Vacía?
Está elemento?

CONSULTORIO

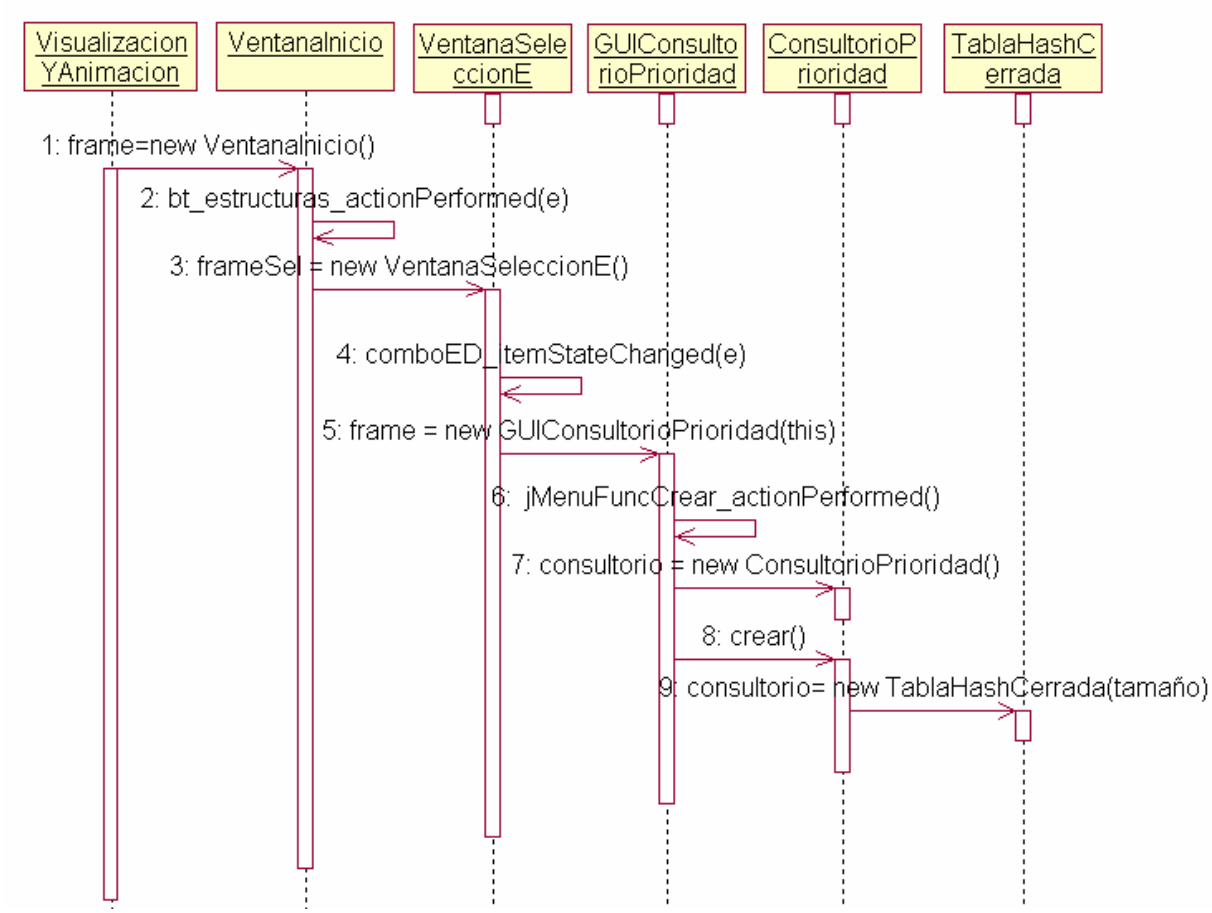


CONSULTORIO PRIORIDAD

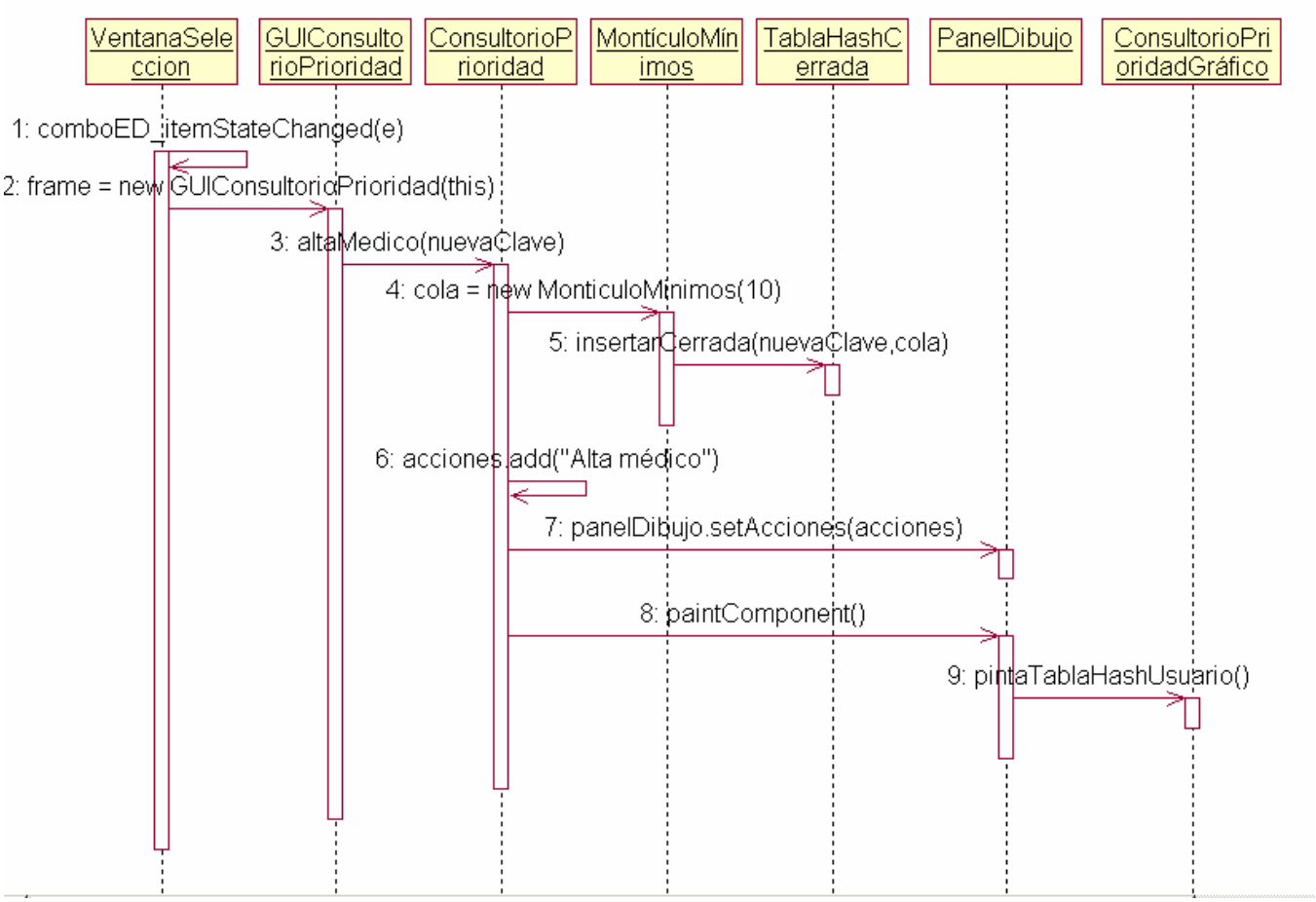


7.2. Diagramas de interacción

Crear un consultorio médico de prioridad



Añadir un nuevo médico al consultorio de prioridad (vista usuario)



8.1. Introducción

Las estructuras de datos y los algoritmos son dos áreas fundamentales en el campo de la informática. En nuestra opinión, esta aplicación resulta una herramienta muy útil para que los alumnos de asignaturas tales como Estructuras de Datos y/o Metodología y Tecnología de la Programación profundicen más en sus conocimientos. Al profesor, también le puede resultar de ayuda a la hora de explicar las materias, como un componente visual e intuitivo en sus explicaciones teóricas.

A continuación se proporciona una ayuda sobre el uso de las ampliaciones realizadas sobre la herramienta para ayudar al usuario a sacar el máximo partido a la misma.

8.2. Ejecución de la Aplicación

8.2.1. Tabla ordenada

Las operaciones disponibles en la ED Tabla ordenada son:

- Crear una tabla ordenada.
- Insertar una pareja (clave, valor)
- Eliminar una pareja.
- Consultar el valor asociado a una clave.
- Consultar si está una clave en la tabla.
- Recorrer ordenada.
- Consultar si la tabla está vacía.

Para aplicar estas funciones, el usuario tiene dos posibilidades:

- Seleccionar la operación desde el menú herramientas → funciones. Con esta opción, para las operaciones que requieran introducir información por parte del usuario (insertar, eliminar, consultar, está) se pasará el control a la caja de funciones y será necesario pulsar el botón aplicar, después de haber introducido los datos necesarios.
- Seleccionar la opción en la caja de funciones. Será necesario pulsar el botón aplicar en todas la operaciones.

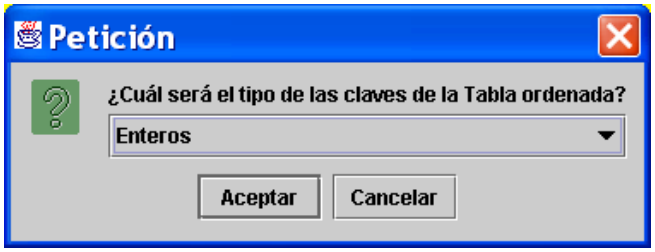
Representación de la Tabla ordenada según las distintas vistas:

- Vista usuario de la herramienta: en esta vista esta estructura de datos se representa con una tabla que consta de dos columnas; la primera es la clave y la segunda el valor. Cada fila representa por tanto una pareja clave, valor. En ella se ven los valores e la tabla ordenados según la clave, que determina unívocamente cada valor asociado.

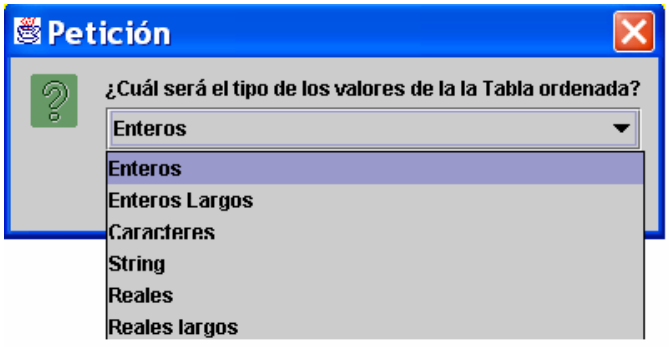
- Vista dinámica: en esta vista se representa mediante un árbol binario de búsqueda cuyos nodos están formados por parejas clave, valor.

Función crear:

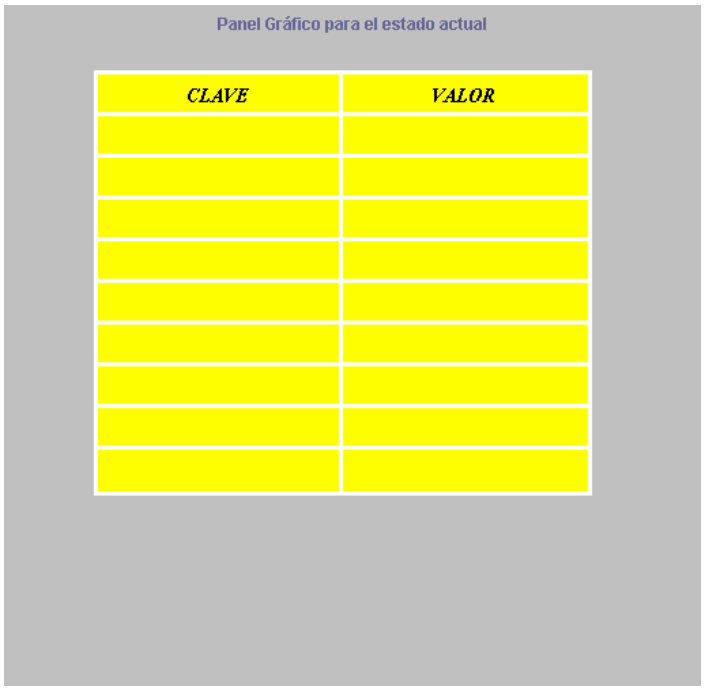
Cuando se crea la tabla ordenada, se piden los tipos de las claves:



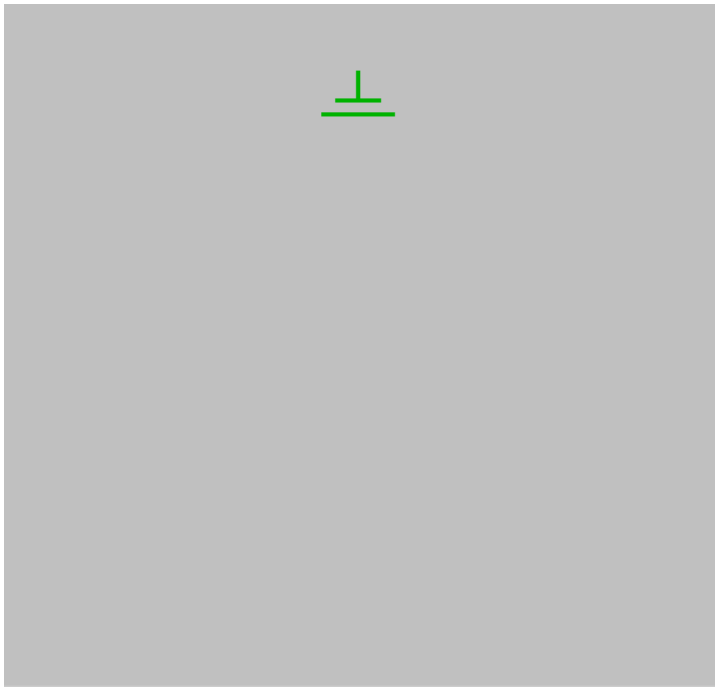
Y de los valores asociados. Hay que hacer notar que el tipo de las claves y de los valores no tiene por qué ser el mismo.



En la vista usuario de la herramienta, la tabla vacía se representa como se muestra en la siguiente figura:



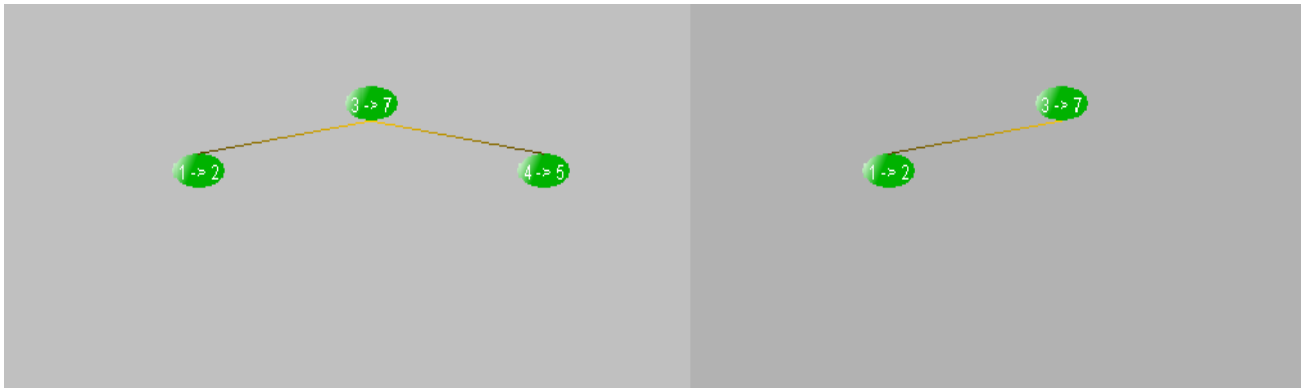
En la vista dinámica, se representa mediante el símbolo de tierra, al igual que se hacía en los árboles:



Función insertar:

A continuación se inserta la pareja (4,5) en la tabla donde ya habían sido insertados los pares (1,2) y (3,7). Los pares se muestran ordenados por orden creciente del valor de la clave. En la vista de usuario se muestra así:

En la vista dinámica se representa así:



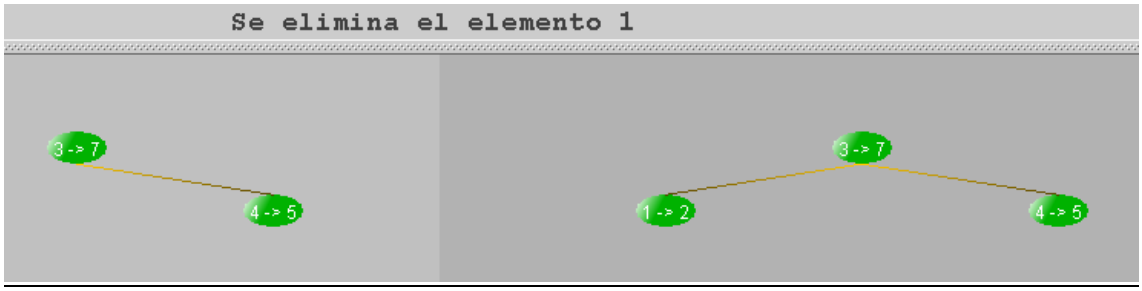
Se recuerda a los usuarios de la aplicación, que el panel de la derecha muestra el estado anterior de la ED.

Hay que mencionar que las claves son únicas, es decir, si insertamos un valor de la clave que ya está en la tabla el programa lanza un error para informar al usuario de lo ocurrido:

Función eliminar:

Se elimina el par cuya clave coincida con la introducida por el usuario. Por ejemplo eliminamos el par (1,2), resultando:

Así mismo en el árbol binario de búsqueda también se elimina.



Función consultar:

Si consultamos el valor que tiene la clave 3, aparece el siguiente mensaje debajo de la barra de menús:

```
El valor del elemento con clave 3 es 7
```

Función está:

Comprobamos si está el elemento de clave 8 en la tabla anterior:

```
El elemento con clave 8 no está en la tabla
```

Sin embargo si repetimos la operación para el 4:

```
El elemento con clave 4 sí está en la tabla
```

Función vacía:

Si la estructura de datos está vacía mostrará el siguiente mensaje:

```
La tabla ordenada está vacía
```

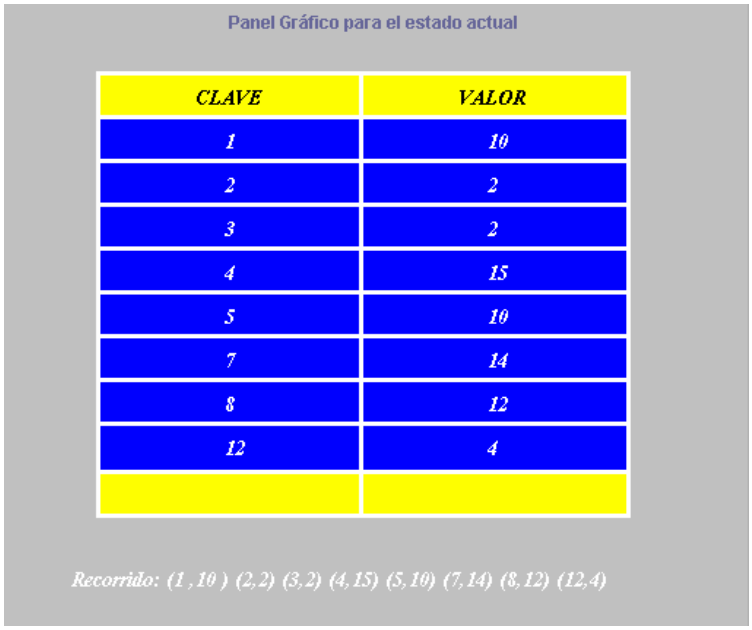
Si no:

```
La tabla ordenada no está vacía
```

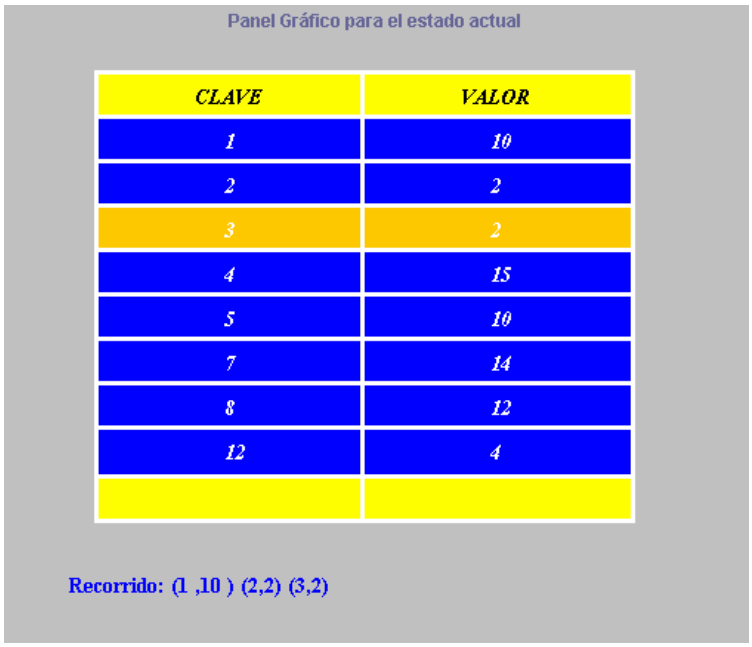
Función recorrer ordenada:

Produce una lista de parejas (clave, valor) ordenadas por clave, como resultado de recorrer en inorden el árbol binario de búsqueda asociado.

Por ejemplo para la siguiente tabla ordenada, el recorrido ordenado de la misma daría como resultado:



En un estado intermedio de la ejecución se puede ver como se van iluminando las filas de la tabla que trata en cada instante si estamos en la vista de usuario, o bien los nodos del árbol si estamos en la vista dinámica:



8.2.2. Tabla hash

Las operaciones disponibles en la ED Tabla Hash son:

- Crear una tabla hash.
- Insertar una pareja (clave, valor)
- Eliminar una pareja por clave.
- Consultar el valor asociado a una clave.
- Consultar si está una clave en la tabla.
- Consultar si la tabla está vacía.

Para aplicar estas funciones, el usuario tiene dos posibilidades:

- Seleccionar la operación desde el menú herramientas → funciones. Con esta opción, para las operaciones que requieran introducir información por parte del usuario (insertar, eliminar, consultar, está) se pasará el control a la caja de funciones y será necesario pulsar el botón aplicar, después de haber introducido los datos necesarios.
- Seleccionar la opción en la caja de funciones. Será necesario pulsar el botón aplicar en todas la operaciones.

Representación de la Tabla Hash según las distintas vistas:

- Vista usuario de la herramienta: en esta vista esta estructura de datos se representa con una tabla que consta de dos columnas; la primera es la clave y la segunda el valor. Cada fila representa por tanto una pareja clave, valor. En ella se ven los valores de la tabla dispuestos sin ningún orden establecido (a diferencia de las tablas ordenadas) y la clave, al ser única, determina unívocamente cada valor asociado.

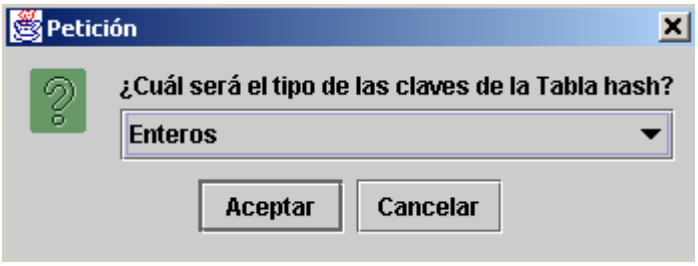
- Vista usuario de desarrollo, implementación tabla abierta: en esta vista se representa la implementación de una tabla hash abierta mediante una columna de valores hash y una lista enlazada de parejas clave-valor asociada a cada valor hash. Se muestra en la parte derecha otra tabla informativa con las operaciones realizadas sobre la tabla, indicando la función hash utilizada, la clave, el valor, el valor hash asociado a la clave y la acción realizada. En este caso se utiliza una función hash que se calcula como la clave módulo del tamaño de la tabla.

- Vista usuario de desarrollo, implementación tabla cerrada: en esta vista se representa la implementación de una tabla hash cerrada mediante una tabla formada por dos columnas, la primera para la clave y la segunda para el valor, de la respectiva pareja. Al eliminar un elemento se muestra de color verde la posición que ha dejado libre, para que el usuario pueda comprobar que será reutilizada en las inserciones siguientes si se produce un adelantamiento de clave.

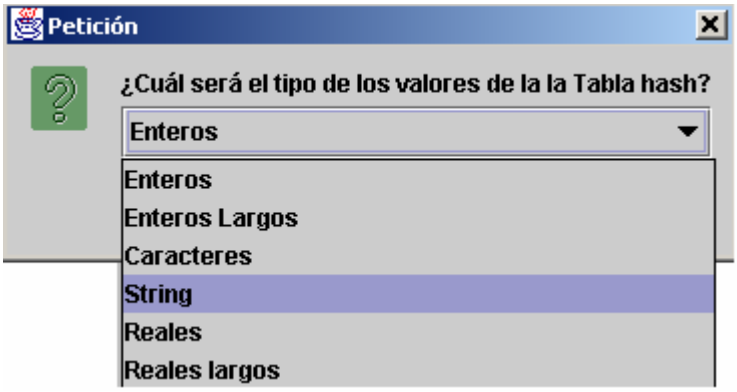
En esta vista se muestra una tabla de información adicional que indica la función hash utilizada, la clave, el valor, el valor hash asociado a la clave, la acción a realizar y, además, información de la situación que se produce: colisión o adelantamiento de clave al insertar. Se utiliza la misma función hash (clave módulo del tamaño de la tabla) y se muestra la función de rehashing que se utiliza cuando se produce una colisión. Se utiliza la función de rehashing cuadrático: (hash anterior + 2*número de colisión-1) módulo del tamaño de la tabla.

Función crear:

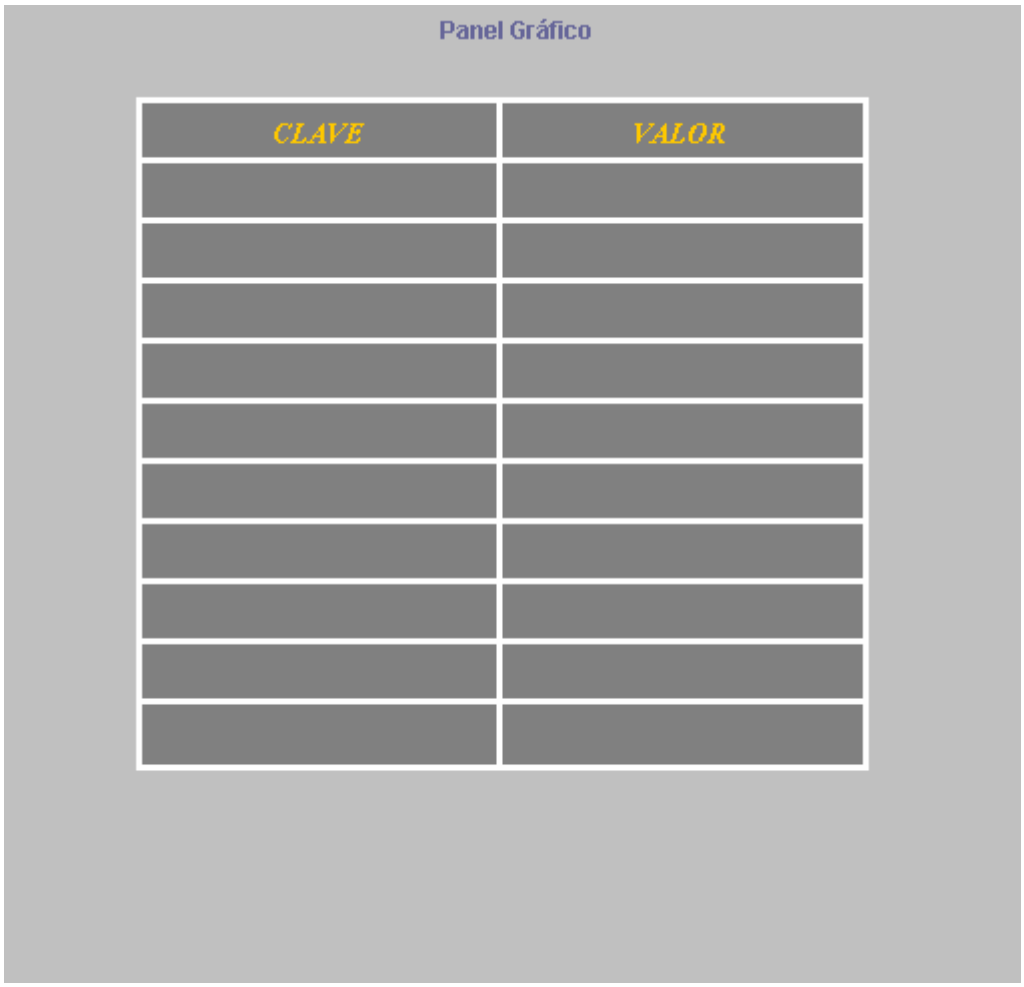
Cuando se crea la tabla hash, se piden los tipos de las claves:



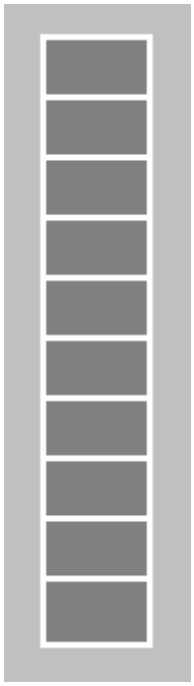
Y de los valores asociados. Hay que hacer notar que el tipo de las claves y de los valores no tiene por qué ser el mismo.



En la vista usuario de la herramienta, la tabla vacía se representa como se muestra en la siguiente figura:



En la vista de la implementación abierta, se representa mediante un vector vacío:



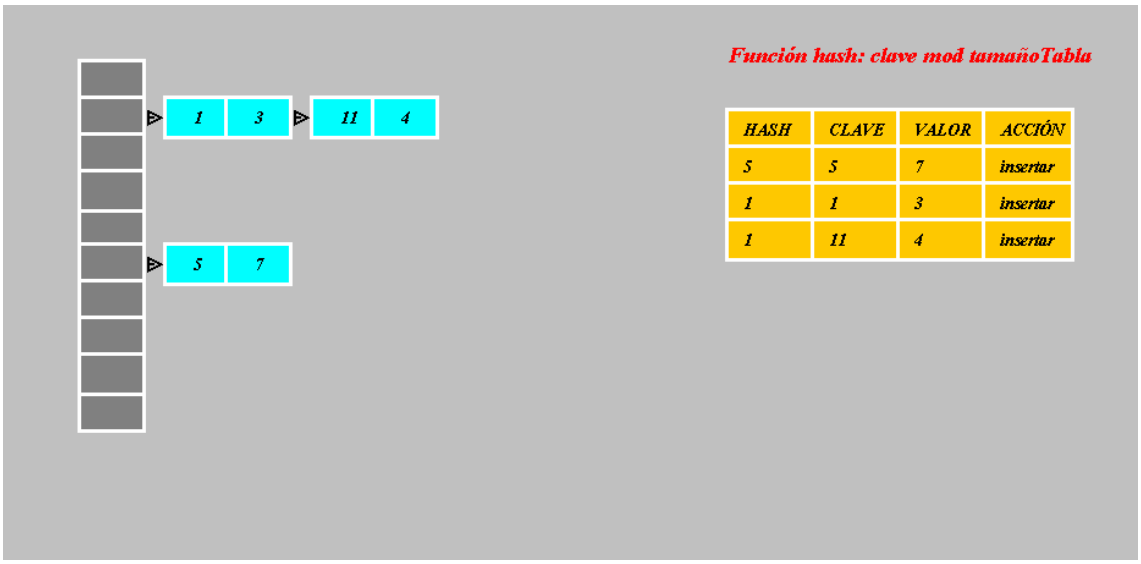
En la vista de la implementación cerrada, la representación de la tabla hash vacía es la misma que en la vista de usuario.

Función insertar:

A continuación se insertan las parejas (5,7), (1,3), (11,4). En la vista de usuario se muestra así:

En esta vista podemos observar como en el panel de la derecha se muestra el estado anterior de la tabla.

En la vista de implementación abierta se representa así:



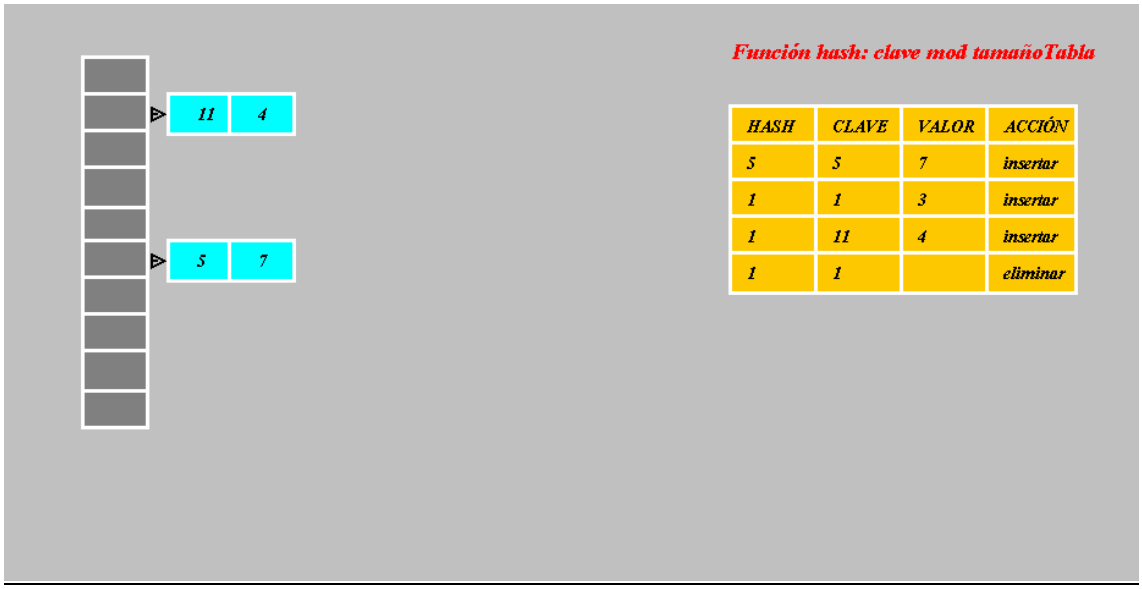
En la vista de implementación cerrada se representa así:

Podemos ver que en estas dos vistas de implementación se informa al usuario sobre las acciones realizadas sobre la tabla y las posibles colisiones que se puedan producir.

Función eliminar:

Se elimina el par cuya clave coincida con la introducida por el usuario. Por ejemplo eliminamos el par (1,3), resultando en la vista de usuario:

Así mismo, el borrado se observa en las vistas de implementación:



CLAVE	VALOR
11	4
5	7

Función hash: clave mod tamañoTabla
Rehashing Cuadrático:
(hash anterior + 2*colisión -1) mod tamañoTabla

HASH	CLAVE	VALOR	ACCIÓN
5	5	7	insertar
1	1	3	insertar
1	11		colisión -> rehash
2	11	4	insertar
1	1		eliminar

Se aprecia como la posición eliminada pasa a tener un color verde. Si ahora insertamos una pareja con una clave que tenga como valor hash asociado el 1, se reutilizará la posición borrada anteriormente en la tabla, mediante adelantamiento de clave:

CLAVE	VALOR
41	6
11	4
5	7

Función hash: clave mod tamañoTabla
Rehashing Cuadrático:
(hash anterior + 2*colisión -1) mod tamañoTabla

HASH	CLAVE	VALOR	ACCIÓN
5	5	7	insertar
1	1	3	insertar
1	11		colisión -> rehash
2	11	4	insertar
1	1		eliminar
2	41		colisión -> rehash
5	41		colisión -> rehash
1	41	6	insertar(adelantar clave)

Función consultar:

Si consultamos el valor que tiene la clave 5, aparece el siguiente mensaje debajo de la barra de menús:

```
El valor del elemento con clave 5 es 7
```

Función está:

Comprobamos si está el elemento de clave 8 en la tabla anterior:

```
El elemento con clave 8 no está en la tabla
```

Sin embargo si repetimos la operación para el 11:

```
El elemento con clave 11 sí está en la tabla
```

Función vacío:

Si la estructura de datos está vacía mostrará el siguiente mensaje:

```
La tabla hash está vacía
```

Si no:

```
La tabla hash no está vacía
```

8.2.3. Consultorios médicos

Operaciones a realizar con el consultorio médico

Se permiten diversas operaciones sobre este tipo abstracto de datos, a saber:

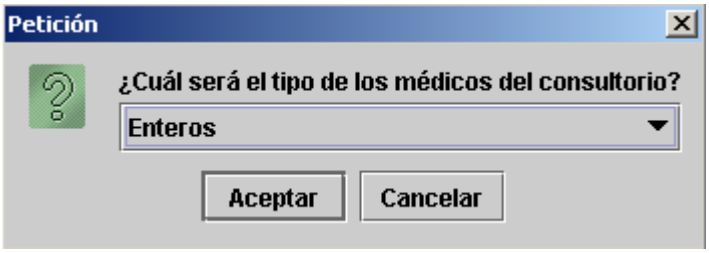
- Crear un consultorio médico.
- Dar de alta un médico.
- Atender una consulta.
- Pedir una consulta a un médico.
- Conocer el siguiente paciente de un médico.
- Saber si tiene pacientes un médico.

A continuación se muestra una interacción realizando todos estos casos para que el usuario aprecie la forma de realizar las operaciones. Con cada operación se mostrará el resultado de aplicarla en cada vista disponible.

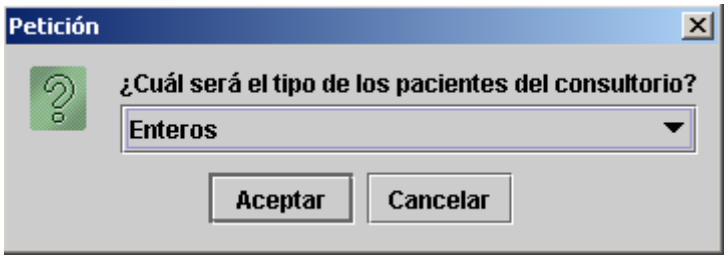
En este caso tenemos dos vistas, la de usuario de la herramienta y la de usuario de desarrollo.

Se comienza creando un consultorio médico, para ello se selecciona la opción de Crear y se pulsa el enter o sobre el botón Aplicar.

Aparece una consulta que nos pide el tipo de médicos del consultorio, un ejemplo sería seleccionar enteros, ahora volvemos a dar al enter o sobre el botón aceptar.

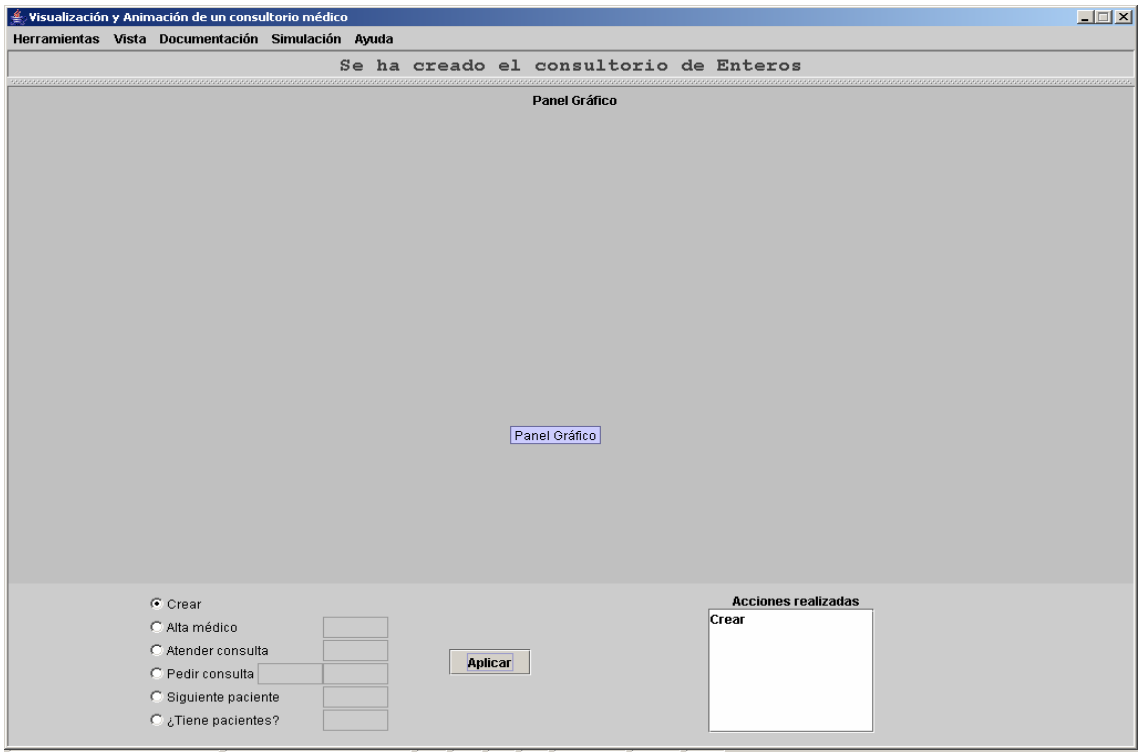


Se nos requiere una nueva información, ahora debemos introducir el tipo de pacientes, repetimos la acción anterior.

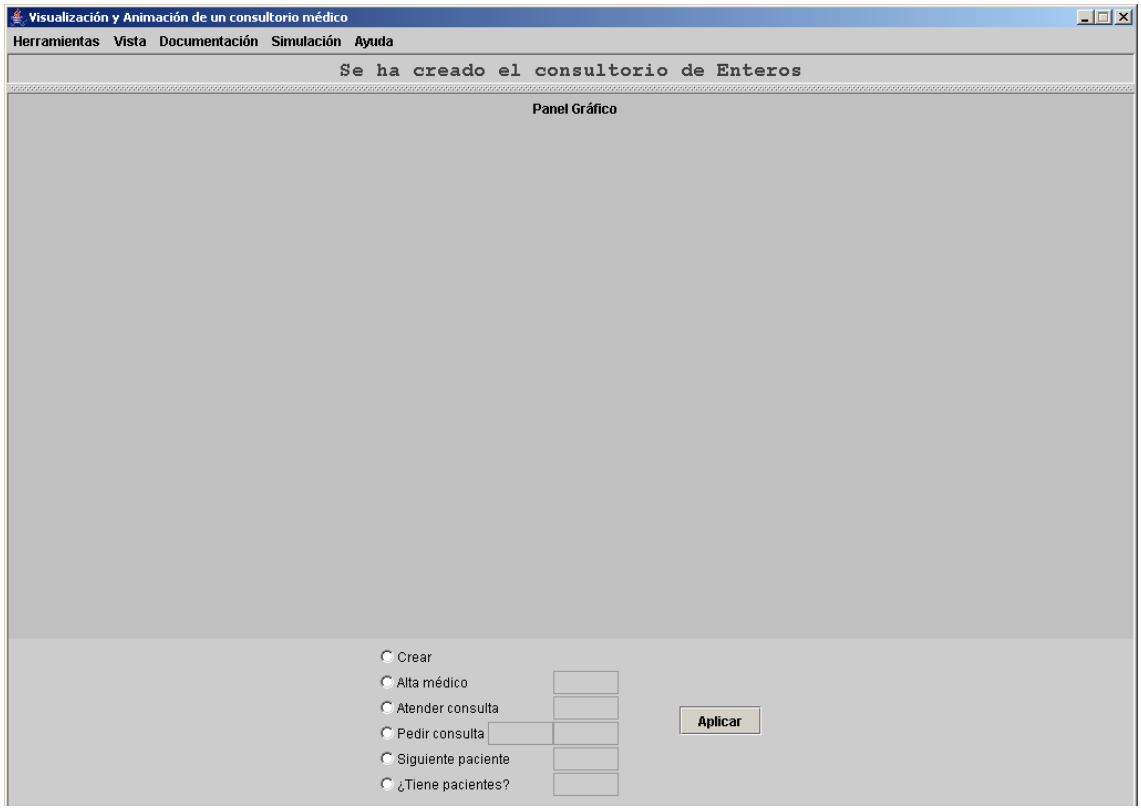


Ya tenemos el consultorio médico creado y el resultado es el siguiente:

Vista de usuario de herramienta

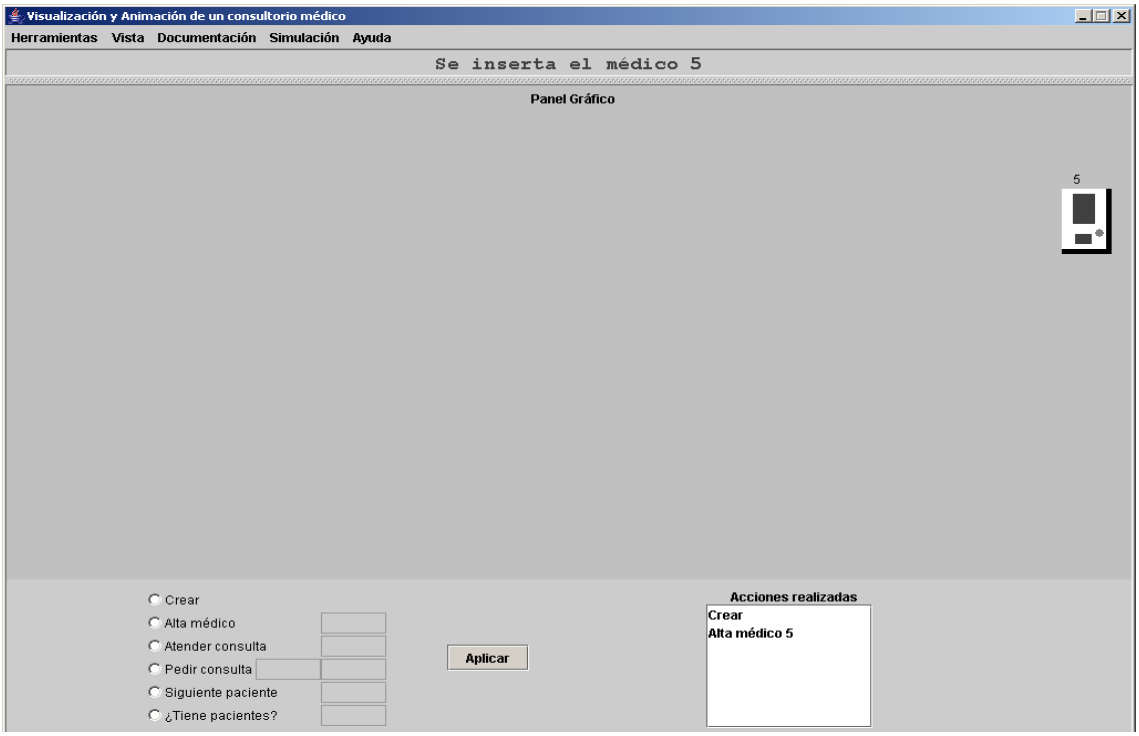


Vista de usuario de desarrollo

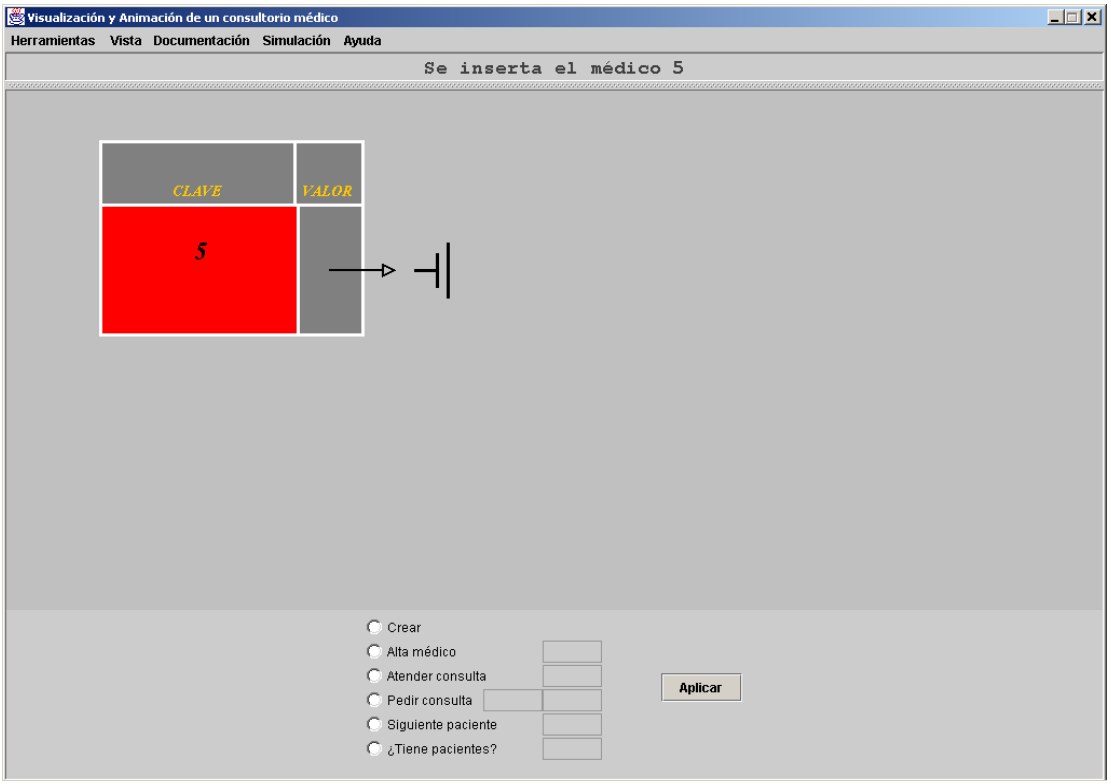


A continuación damos de alta a un médico; esto se consigue de la siguiente manera, seleccionamos la opción de Alta médico e introducimos un valor del tipo seleccionado anteriormente en el campo de texto que se nos acaba de habilitar, por ejemplo 5, ahora pulsamos enter o marcamos el botón Aplicar. Un nuevo médico se nos habrá creado, pudiéndose apreciar la creación de la puerta de su consultorio.

Vista de usuario de herramienta

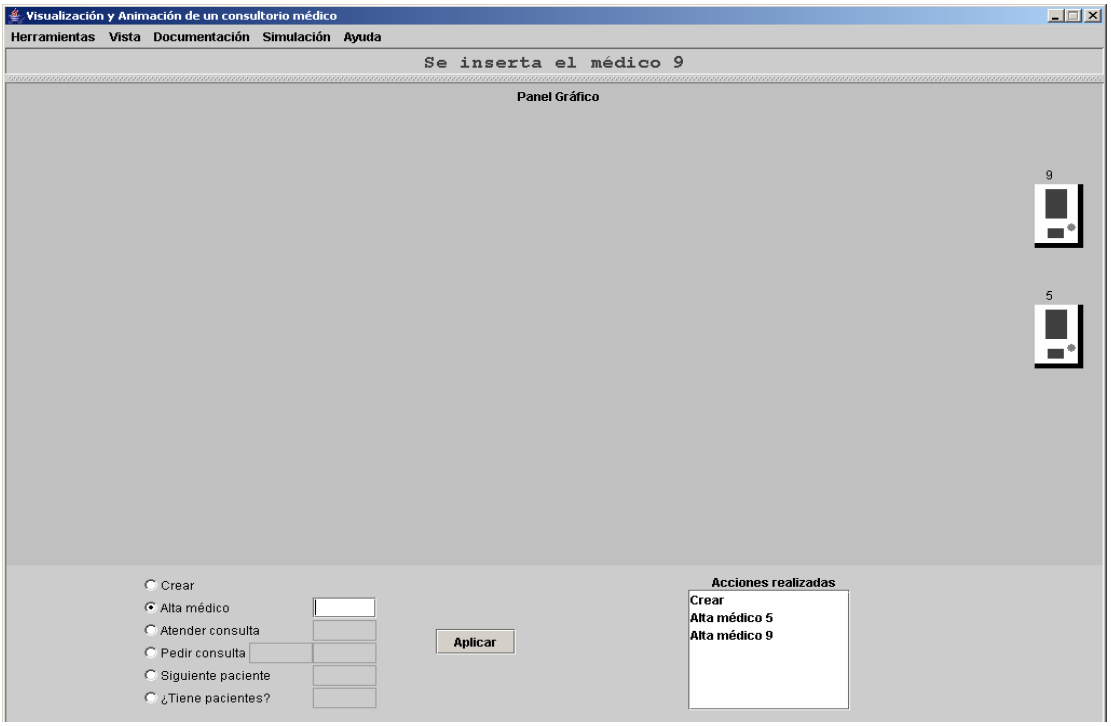


Vista de usuario de desarrollo

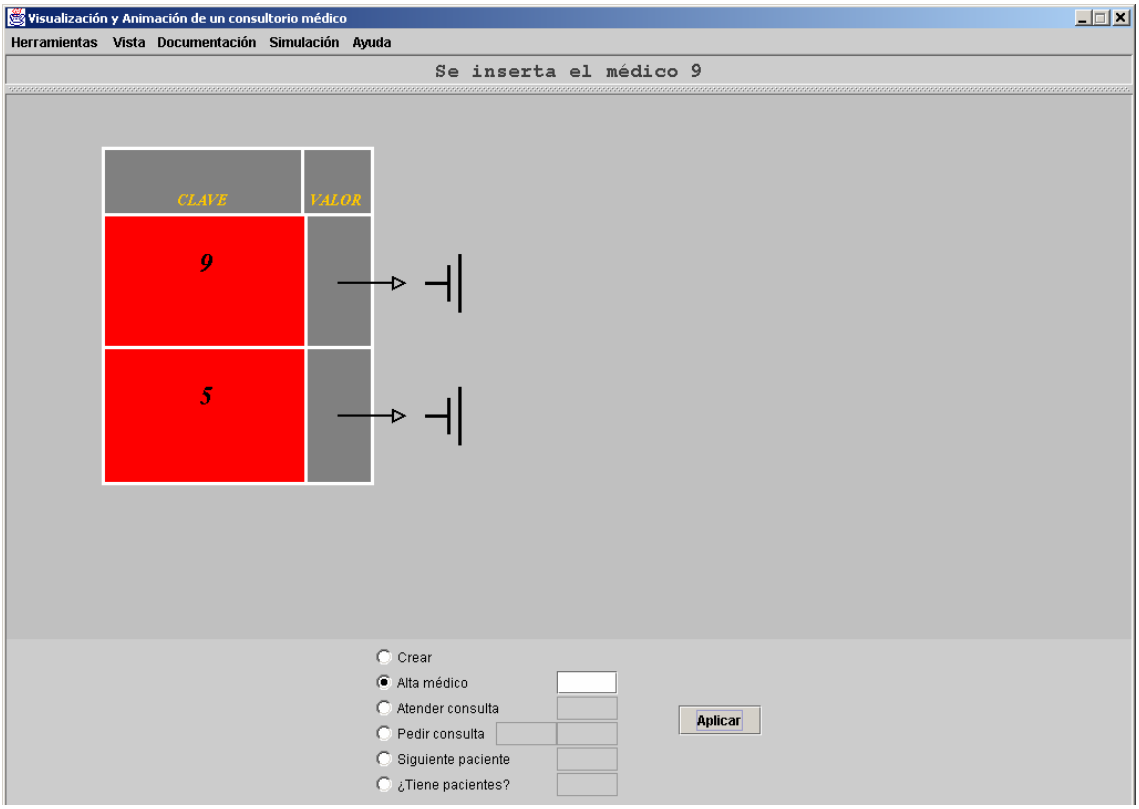


Para dar de alta otro médico se debe repetir la secuencia anterior, teniendo en cuenta que no se puede repetir la clave del médico. Damos de alta el médico 9.

Vista de usuario de herramienta



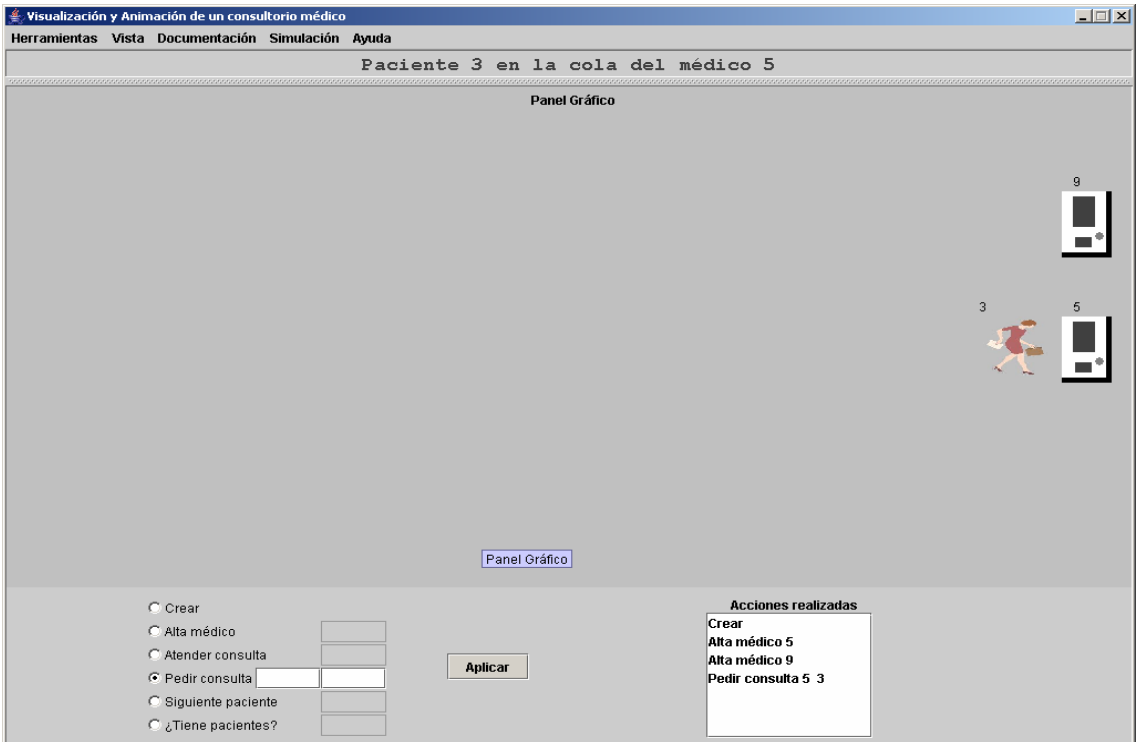
Vista de usuario de desarrollo



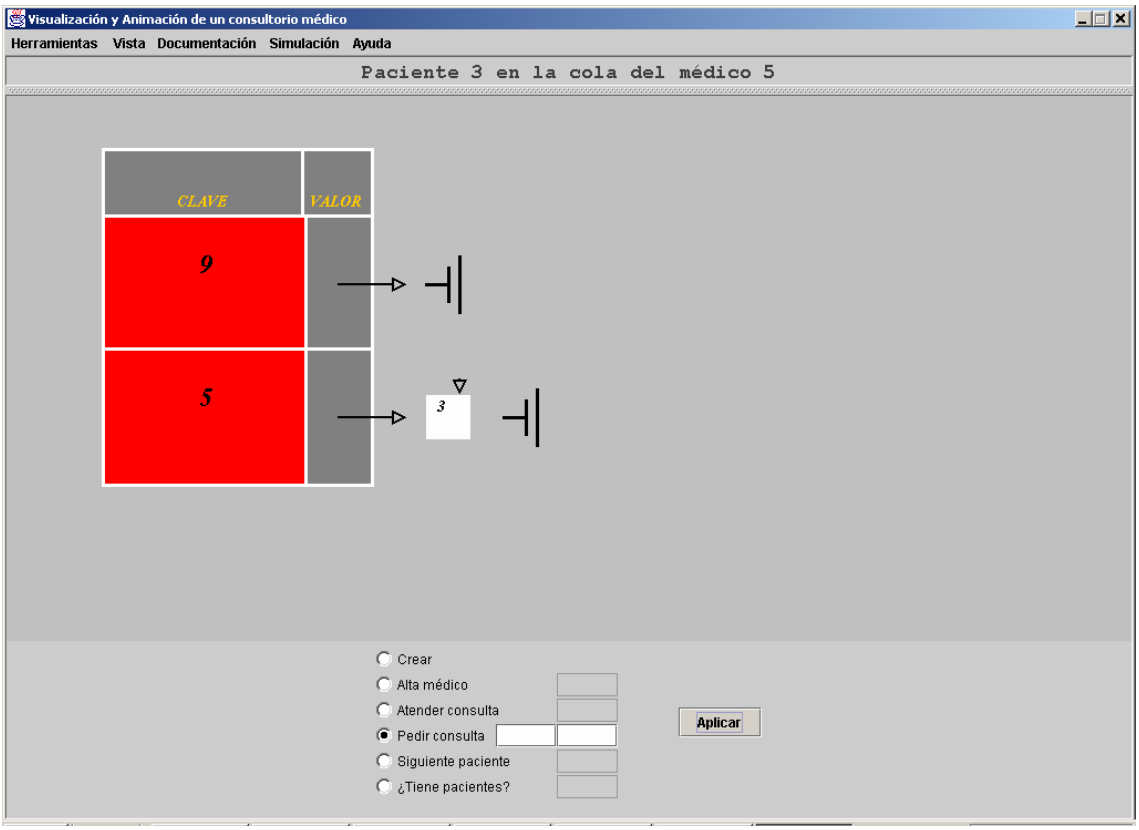
Por limitaciones de visualización, el número máximo de médicos que se pueden tener en el consultorio es de tres.

Si queremos Pedir una consulta para el médico que acabamos de dar de alta, lo que se tiene que hacer es marcar la opción de Pedir consulta e introducir en el campo de texto de la izquierda la clave del médico en el que queremos pedir cita, 5, y en el derecho el valor del paciente, por ejemplo 3. Tras esto pulsamos enter o marcamos el botón Aplicar. Ya hemos pedido consulta.

Vista de usuario de herramienta

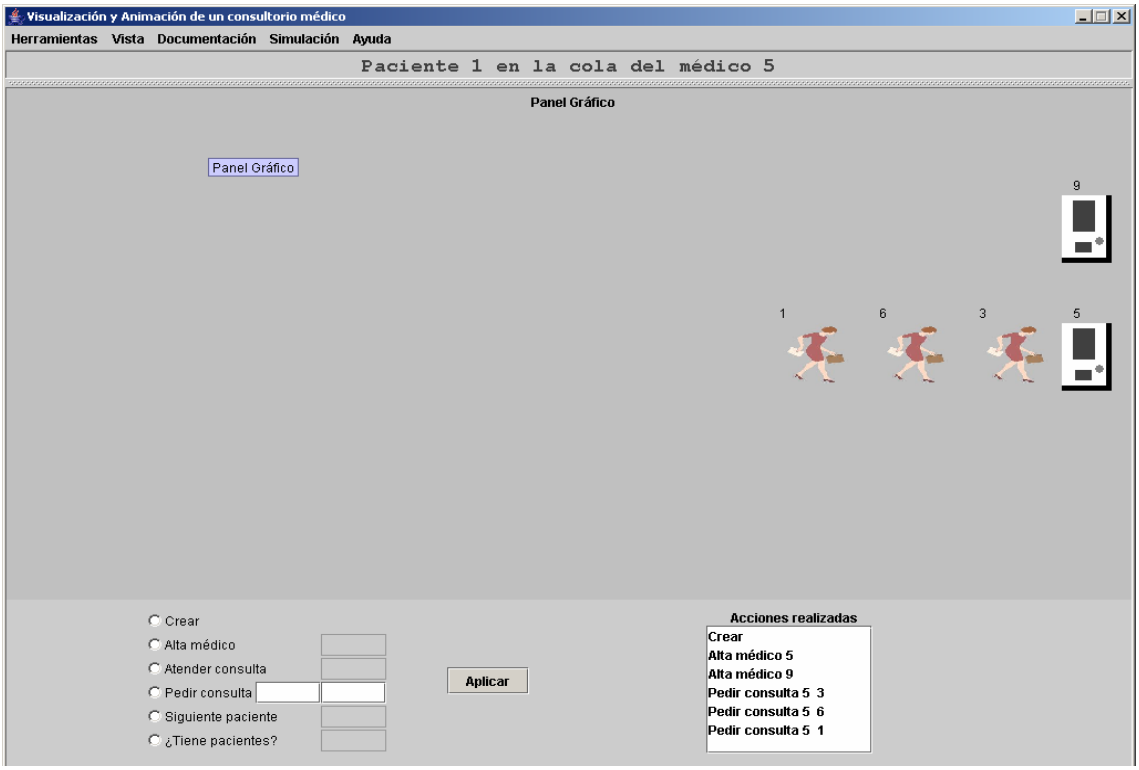


Vista de usuario de desarrollo

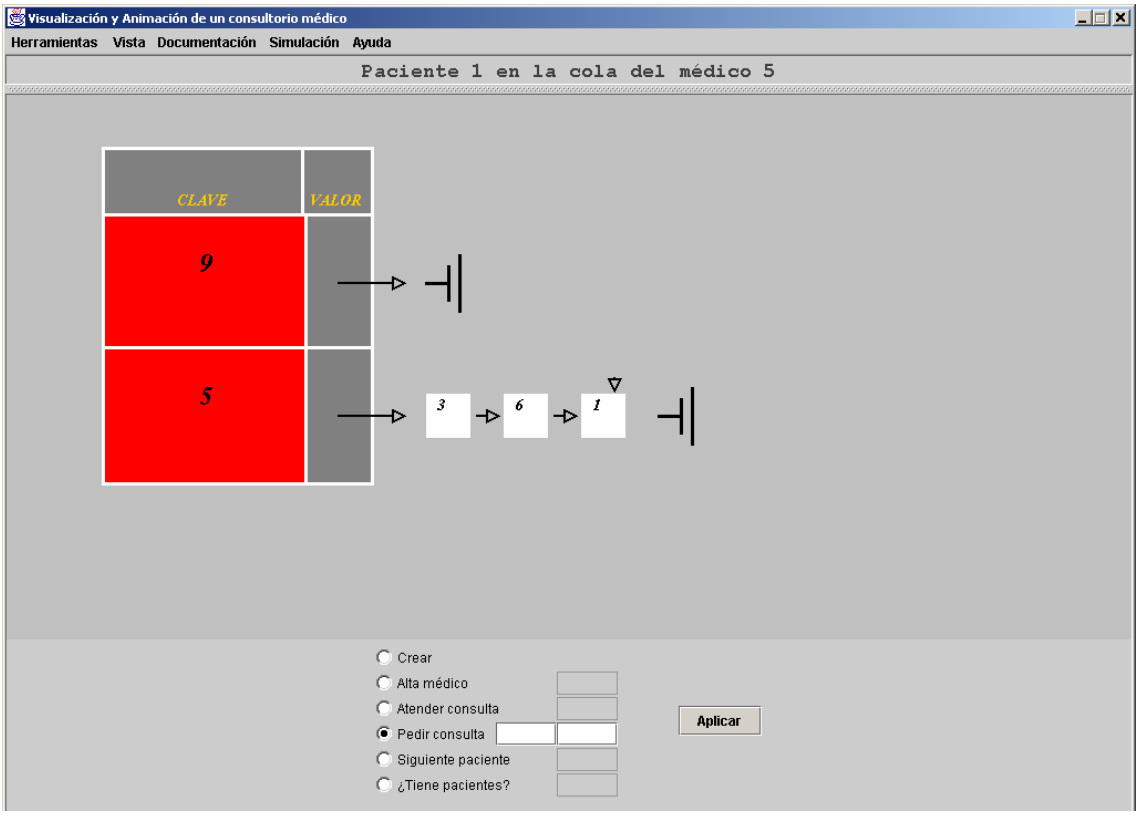


Se realiza la misma secuencia de acciones para añadir nuevos pacientes a ese médico. Se han añadido los pacientes 6 y 1.

Vista de usuario de herramienta



Vista de usuario de desarrollo



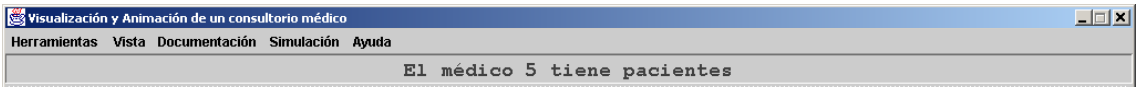
Para saber el siguiente paciente de ese médico se marca la opción de siguiente paciente, se introduce la clave del médico y se pulsa enter o se marca el botón Aplicar. Se muestra en la parte superior de la ventana el siguiente paciente de ese médico. En este ejemplo el siguiente paciente es el 3.Vista de usuario de herramienta

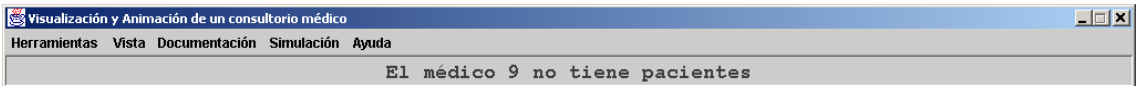
Vista de usuario de desarrollo

Si lo que se pretende es saber el si tiene pacientes un médico, se marca la opción de siguiente paciente, se introduce la clave del médico y se pulsa enter o se marca el botón Aplicar. Se muestra en la parte superior de la ventana el siguiente paciente de ese médico. En caso de consultar el médico 5 la respuesta será que sí tiene pacientes; sin embargo, si consultamos el médico 9 la respuesta será negativa.

Vista de usuario de herramienta

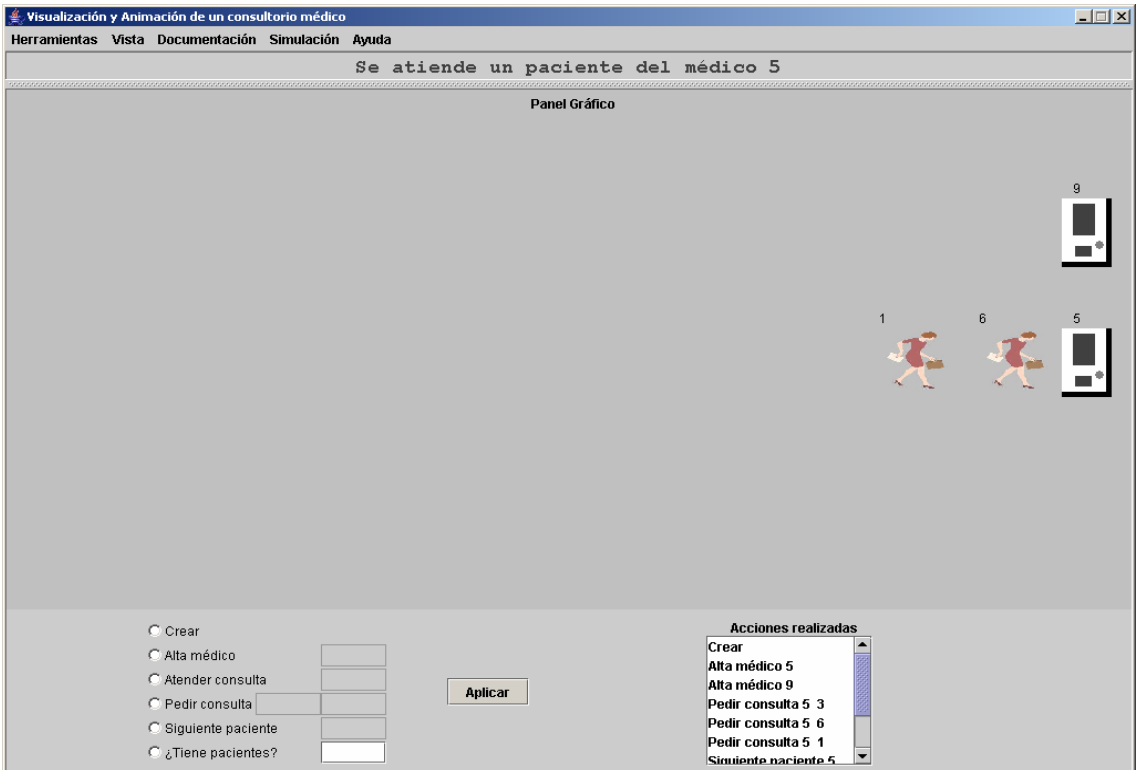
Vista de usuario de desarrollo



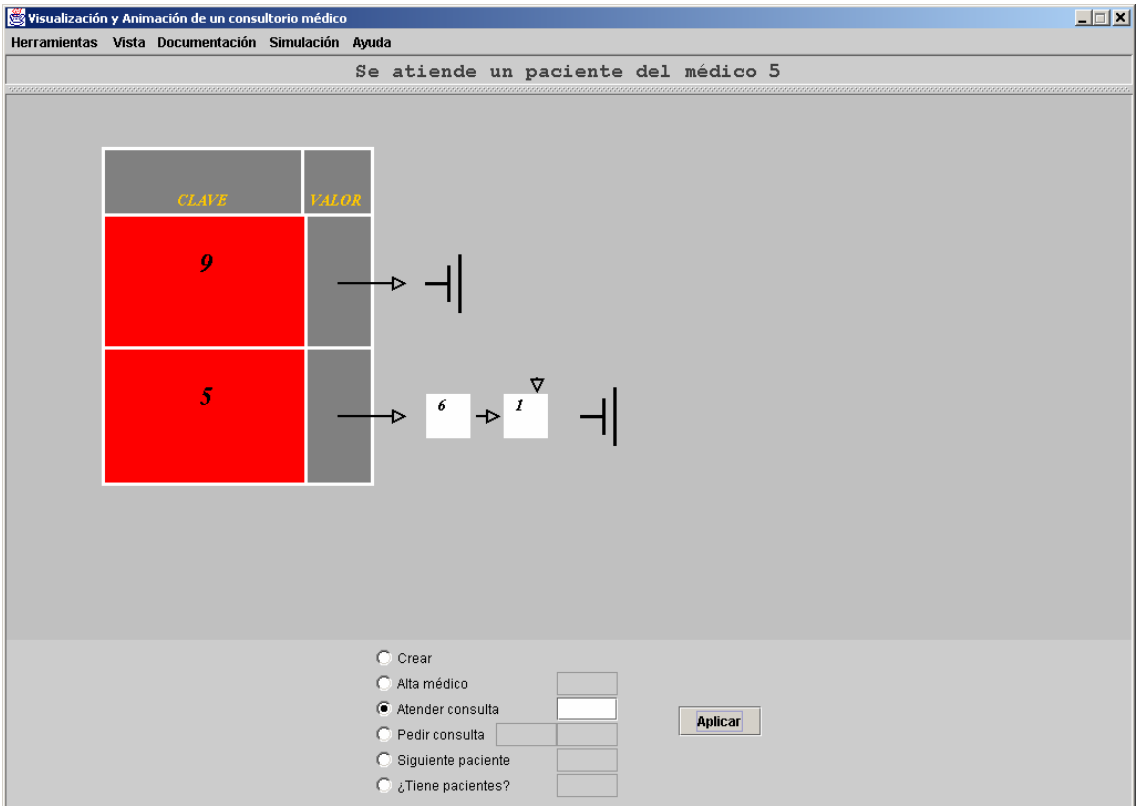


Sólo falta por realizar la operación de atender consulta, para ello seleccionamos esta opción e introducimos el médico que queremos que atienda consulta, el 5. Se pulsa enter o se marca el botón Aplicar. Se elimina el paciente 3.

Vista de usuario de herramienta



Vista de usuario de desarrollo



Consultorio médico con prioridad

Operaciones a realizar con el consultorio médico con prioridad

Se permiten diversas operaciones sobre este tipo abstracto de datos, a saber:

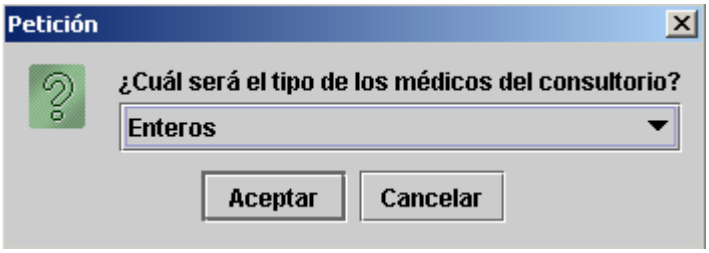
- Crear un consultorio médico con prioridad.
- Dar de alta un médico.
- Atender una consulta.
- Pedir una consulta a un médico.
- Conocer el siguiente paciente de un médico.
- Saber si tiene pacientes un médico.

A continuación se muestra una interacción realizando todos estos casos para que el usuario aprecie la forma de realizar las operaciones. Con cada operación se mostrará el resultado de aplicarla en cada vista disponible.

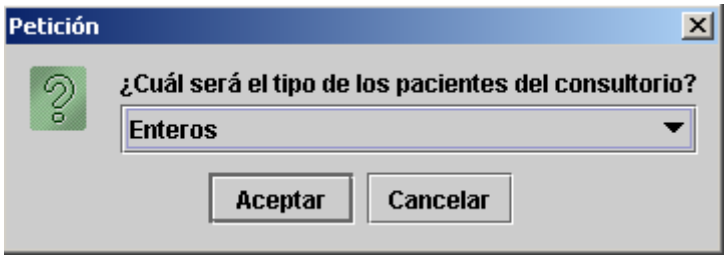
En este caso tenemos dos vistas, la de usuario de la herramienta y la de usuario de desarrollo.

Se comienza creando un consultorio médico con prioridad, para ello se selecciona la opción de Crear y se pulsa el enter o sobre el botón Aplicar.

Aparece una consulta que nos pide el tipo de médicos del consultorio, un ejemplo sería seleccionar enteros, ahora volvemos a dar al enter o sobre el botón aceptar.

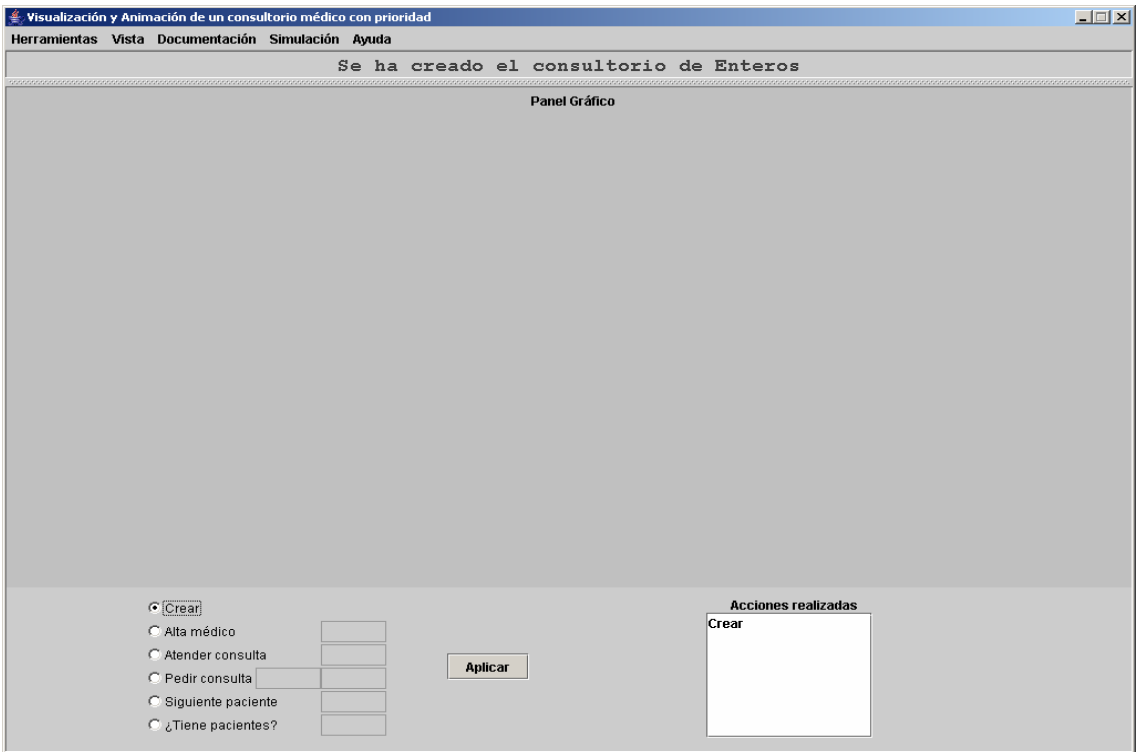


Se nos requiere una nueva información, ahora debemos introducir el tipo de pacientes, repetimos la acción anterior.

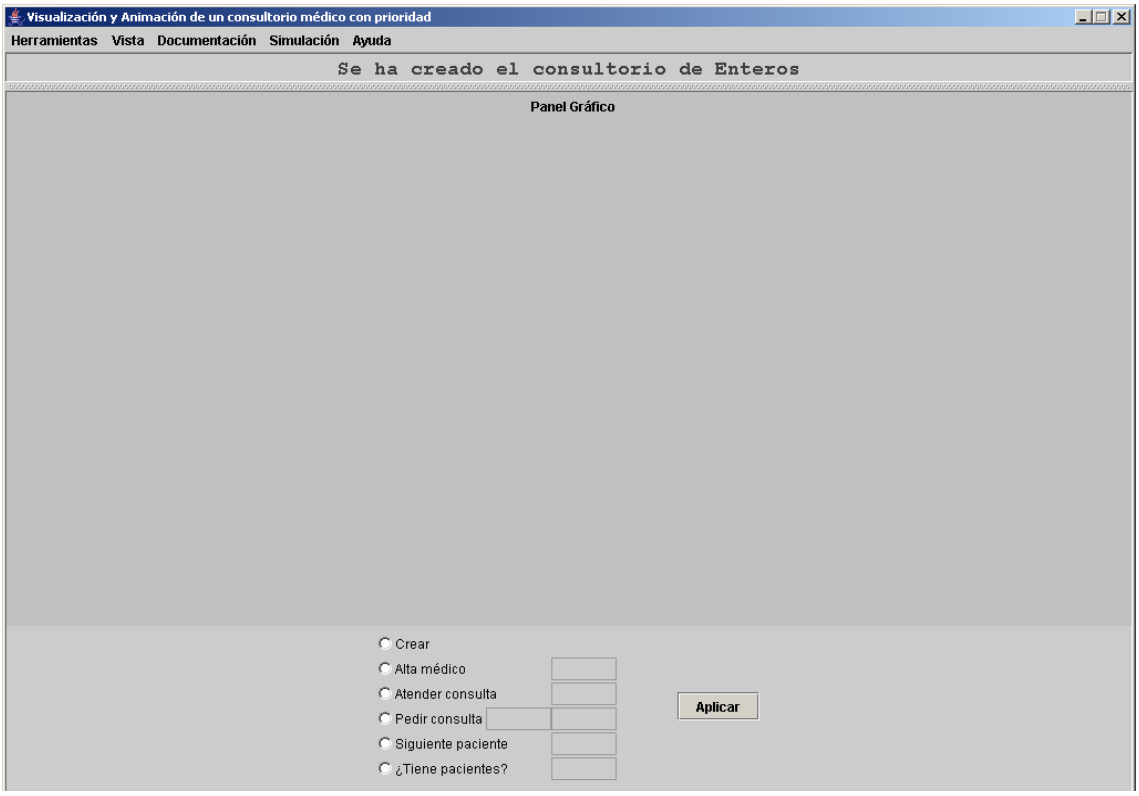


Ya tenemos el consultorio médico creado y el resultado es el siguiente:

Vista de usuario de herramienta

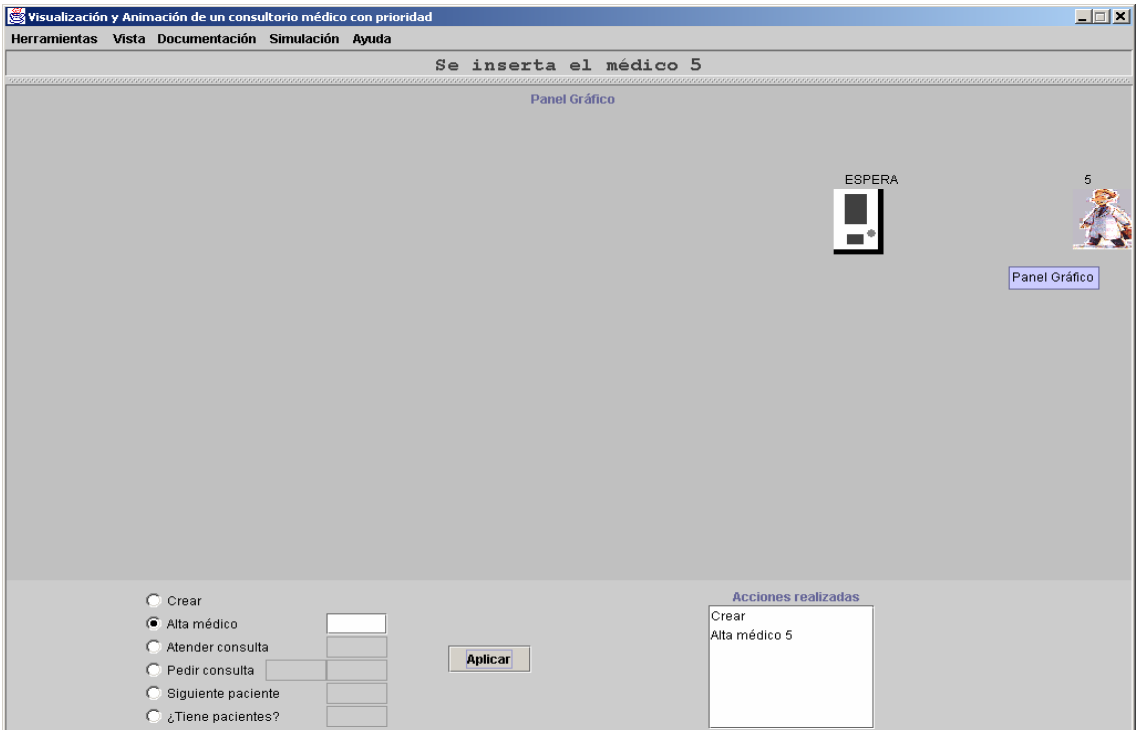


Vista de usuario de desarrollo

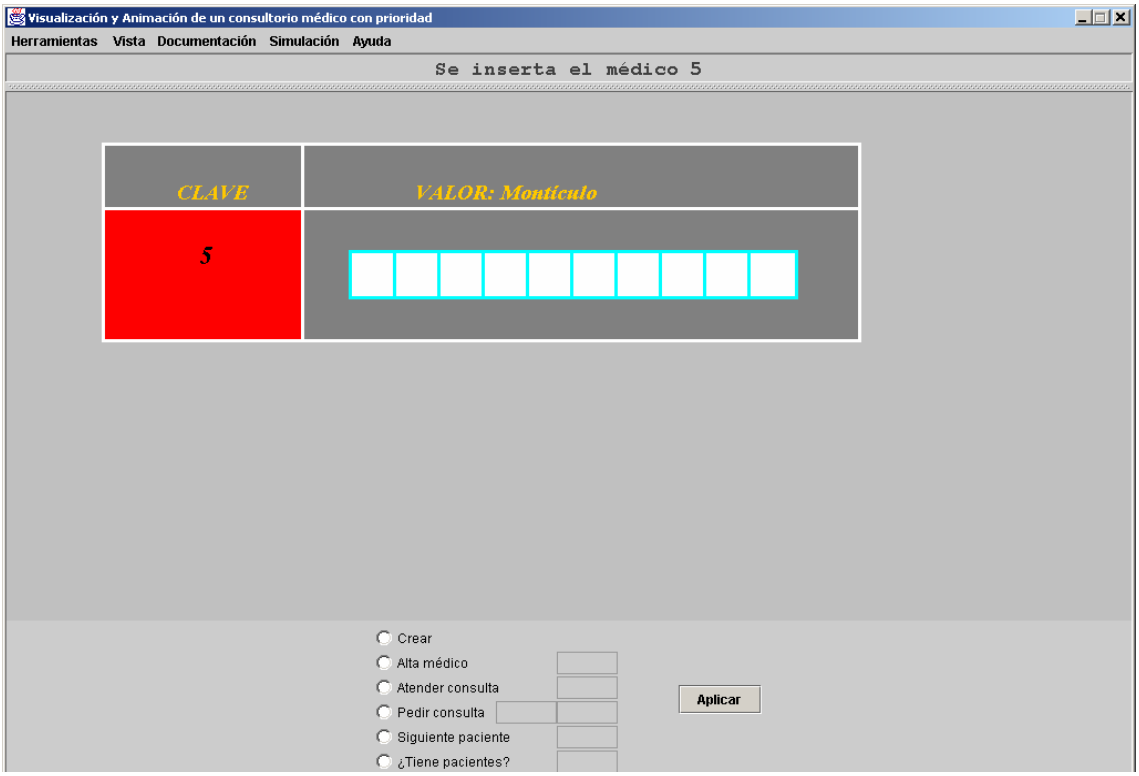


A continuación damos de alta a un médico; esto se consigue de la siguiente manera, seleccionamos la opción de Alta médico e introducimos un valor del tipo seleccionado anteriormente en el campo de texto que se nos acaba de habilitar, por ejemplo 5, ahora pulsamos enter o marcamos el botón Aplicar. Un nuevo médico se nos habrá creado, pudiéndose observar el nuevo médico y la puerta de su sala de espera.

Vista de usuario de herramienta



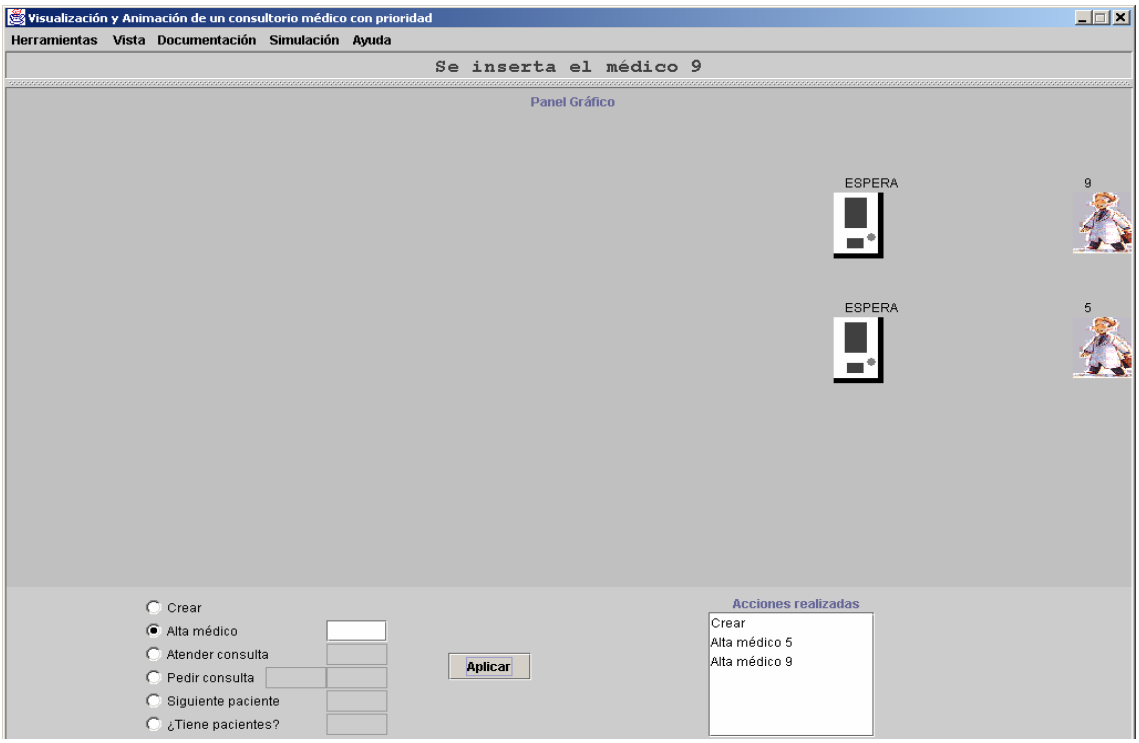
Vista de usuario de desarrollo



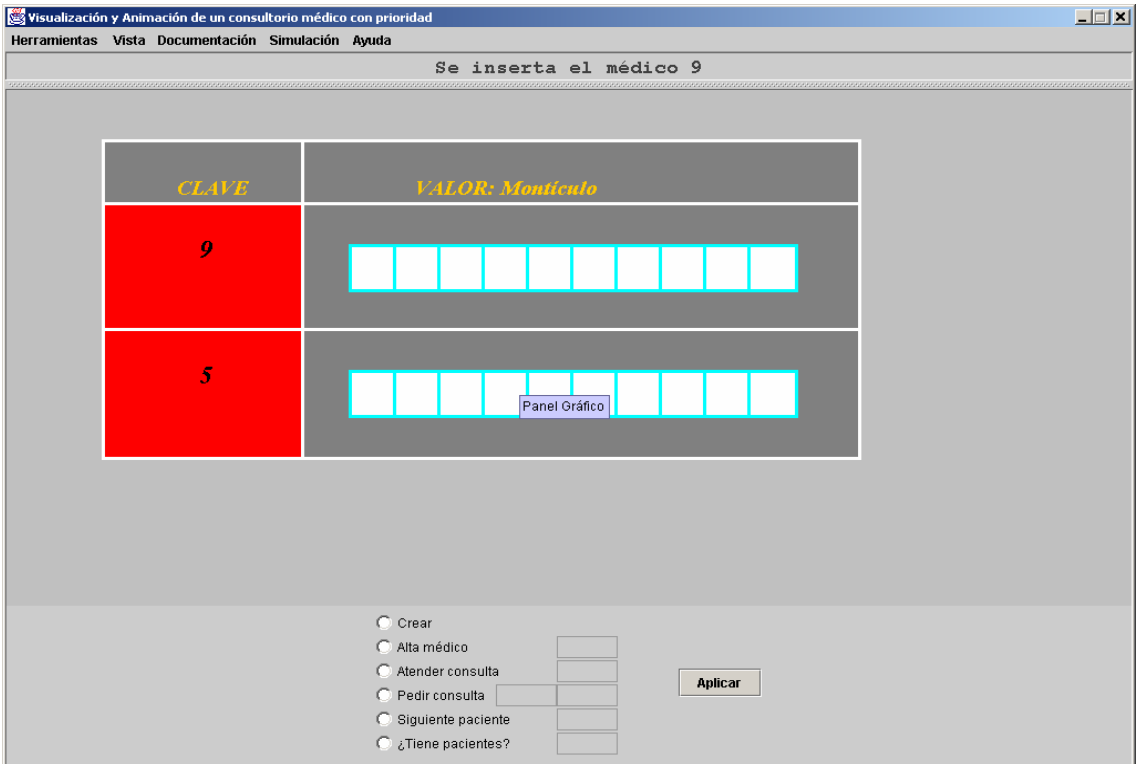
En esta imagen se observa el un vector correspondiente a los pacientes de médico 5. Mencionar que este vector tiene un tamaño estático de tamaño 10 y es implementado mediante un montículo de mínimos.

Para dar de alta otro médico se debe repetir la secuencia anterior, teniendo en cuenta que no se puede repetir la clave del médico. Damos de alta el médico 9.

Vista de usuario de herramienta

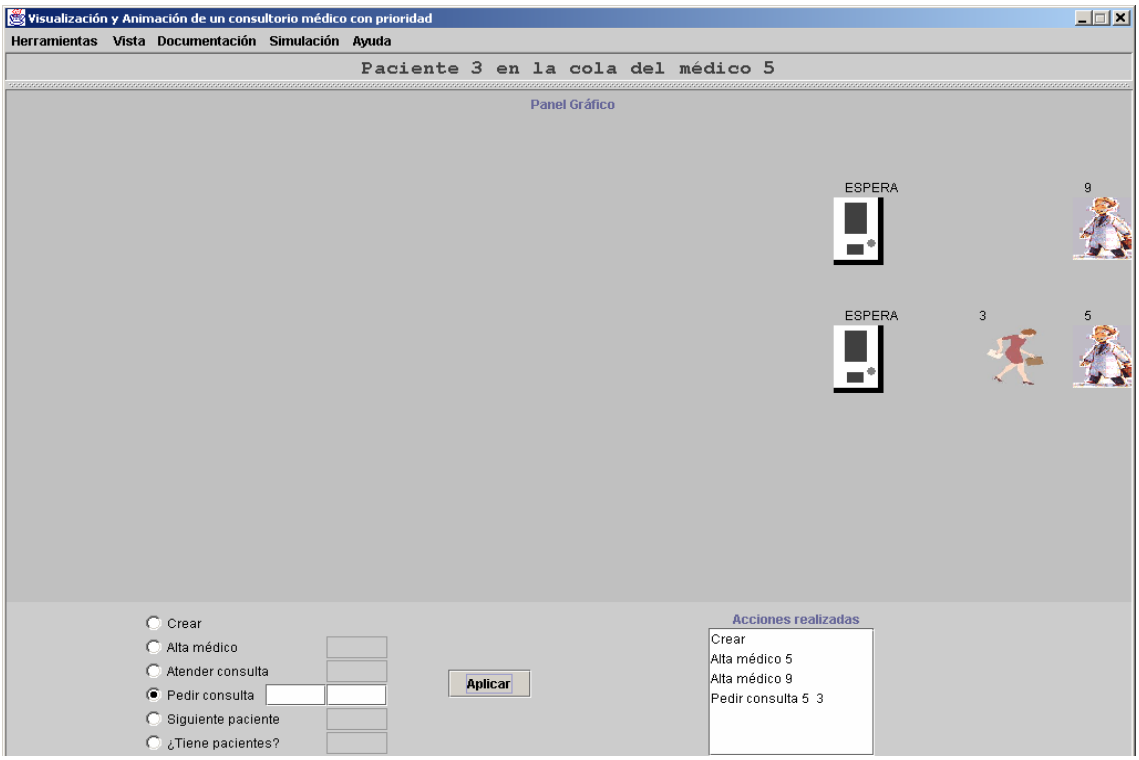


Vista de usuario de desarrollo

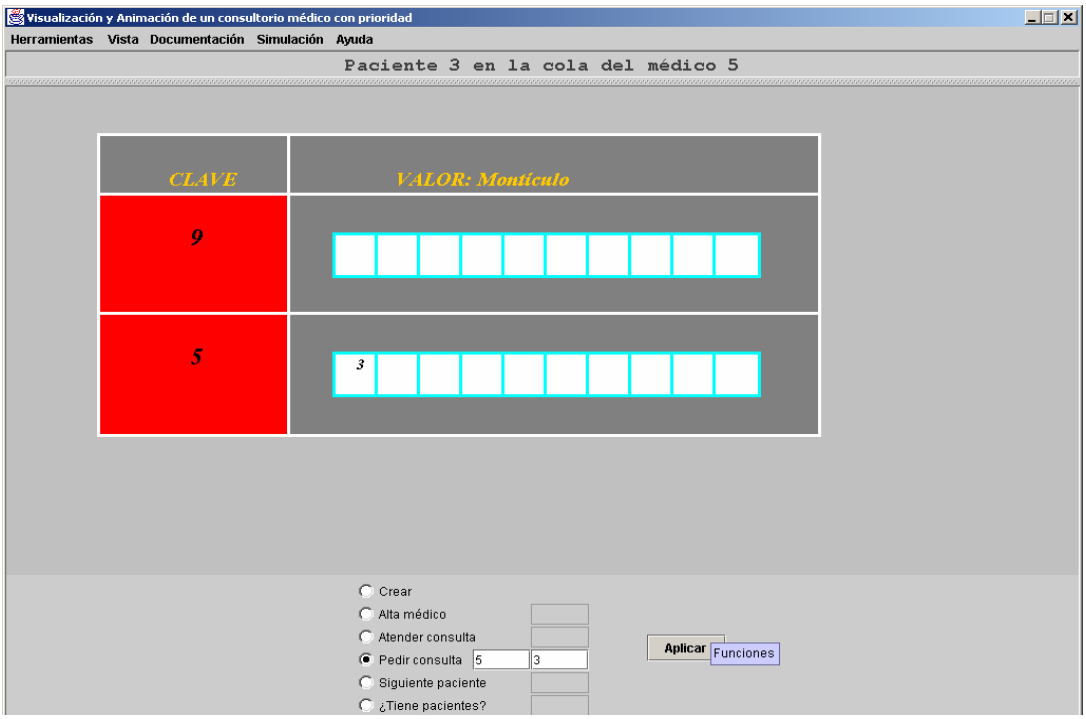


Si queremos Pedir una consulta para el médico que acabamos de dar de alta, lo que se tiene que hacer es marcar la opción de Pedir consulta e introducir en el campo de texto de la izquierda la clave del médico en el que queremos pedir cita, 5, y en el derecho el valor del paciente, por ejemplo 3. Tras esto pulsamos enter o marcamos el botón Aplicar. Ya hemos pedido consulta.

Vista de usuario de herramienta

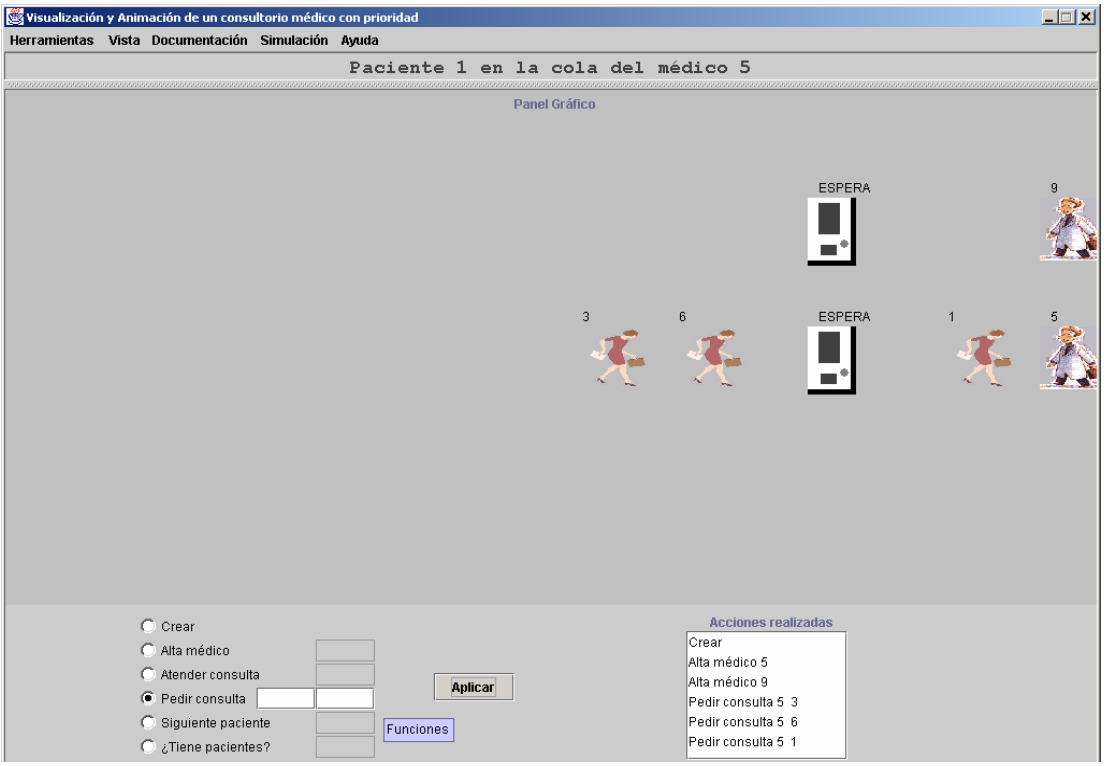


Vista de usuario de desarrollo

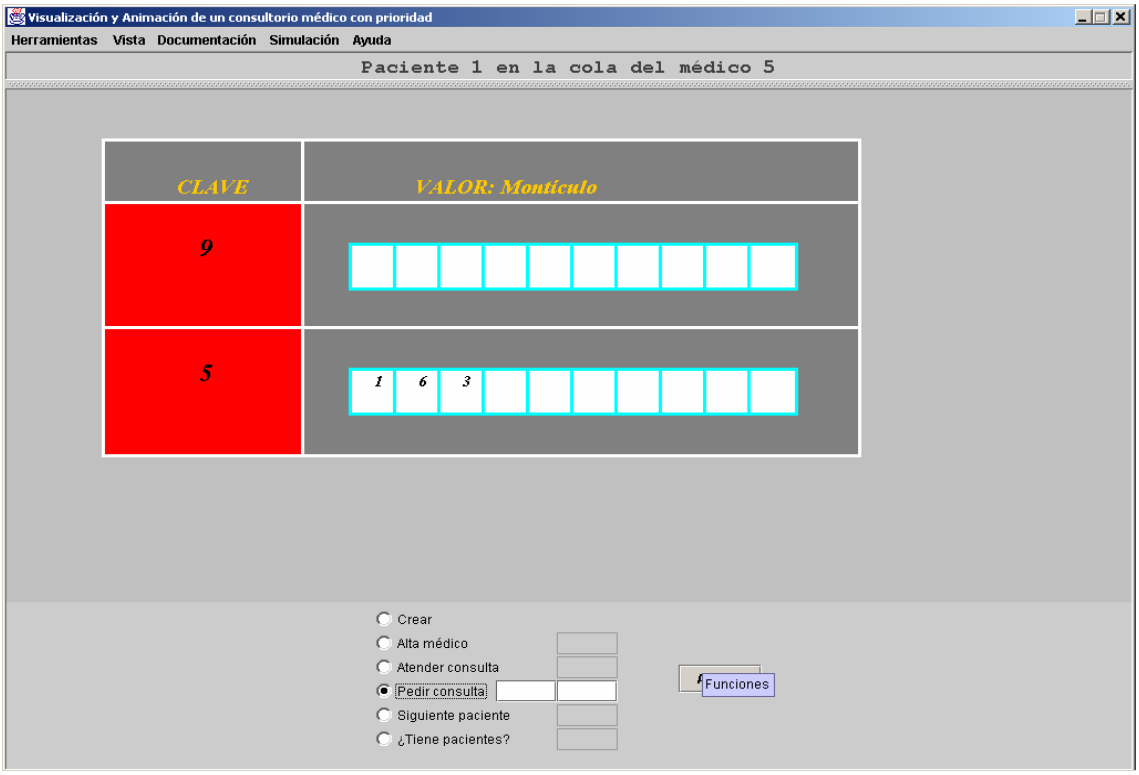


Se realiza la misma secuencia de acciones para añadir nuevos pacientes a ese médico. Se han añadido los pacientes 6 y 1. Destacar que se añaden en función de su prioridad.

Vista de usuario de herramienta



Vista de usuario de desarrollo



Para saber el siguiente paciente de ese médico se marca la opción de siguiente paciente, se introduce la clave del médico y se pulsa enter o se marca el botón Aplicar. Se muestra en la parte superior de la ventana el siguiente paciente de ese médico. En este ejemplo el siguiente paciente es el 1.

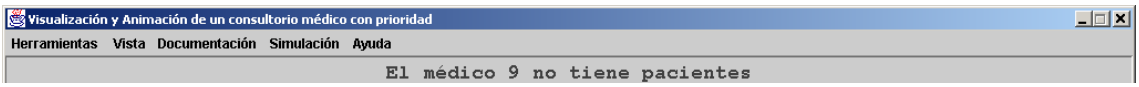
Vista de usuario de herramienta

Vista de usuario de desarrollo

Si lo que se pretende es saber el si tiene pacientes un médico, se marca la opción de siguiente paciente, se introduce la clave del médico y se pulsa enter o se marca el botón Aplicar. Se muestra en la parte superior de la ventana el siguiente paciente de ese médico. En caso de consultar el médico 5 la respuesta será que sí tiene pacientes; sin embargo, si consultamos el médico 9 la respuesta será negativa.

Vista de usuario de herramienta

Vista de usuario de desarrollo



Sólo falta por realizar la operación de atender consulta, para ello seleccionamos esta opción e introducimos el médico que queremos que atienda consulta, el 5. Se pulsa enter o se marca el botón Aplicar. Se elimina el paciente 1.

Vista de usuario de herramienta

Vista de usuario de desarrollo

MANTENIMIENTO DE LA HERRAMIENTA

La herramienta que se entrega es un producto estable y funciona correctamente, sin embargo, es mejorable y ampliable. A continuación se explica la manera de hacerlo.

Mantenimiento Perfectivo

El sistema actual está preparado para ser ampliado de una manera sencilla. Para realizarlo hay que seguir los siguientes pasos:

1. Añadir a las ventanas de selección la estructura que se desea implementar. La mejor forma de hacer esto es copiar una llamada existente y modificarla añadiendo el nombre la nueva estructura.
2. Crear las clases correspondientes a las operaciones de la estructura, GUI de la estructura, estructura Gráfica.

   Con el fin de mantener la estructura de la aplicación se recomienda partir de una interfaz ya existente y modificarla.
3. Añadir a la clase Panel Dibujo la llamada correspondiente a la parte gráfica de la estructura. Tomar una de las existentes como ejemplo y hacer las mismas operaciones para la nueva estructura, de esta forma se conseguirá un código homogéneo.

Mantenimiento Adaptativo

La herramienta funciona actualmente en las versiones de Windows 98/2000/ME/XP, pero no se garantiza su correcto funcionamiento en el resto de sistemas operativos. Una posible mejora en este sentido sería probarlo en otros sistemas y verificar su funcionamiento.

Mantenimiento Correctivo

El sistema realizado ha sido probado y analizado en profundidad, a pesar de esto, es posible que existan fallos. Una mejora consistiría en parchear estos fallos.

Mantenimiento Preventivo

Este tipo de mantenimiento lo hemos realizado este año al modularizar la antigua clase de Interfaz. Siguiendo por este camino se debería modularizar la clase PanelDibujo, creando un panel para cada estructura y facilitando de este modo las futuras ampliaciones.

VALORACIÓN DEL TRABAJO REALIZADO

La ampliación de la presente herramienta nos ha dado la oportunidad de participar en un proyecto ya iniciado, con las dificultades que esto conlleva: trabajar sobre un código ajeno, con una estructura fija que había que respetar para mantener la coherencia con lo que había ya implementado.

El sistema desarrollado es un sistema complementario al sistema de enseñanza actual. Es útil para profesores y alumnos ya que ayuda a comprender mejor las estructuras de datos y los métodos algorítmicos de una forma amena y motivadora.

Las extensiones realizadas permiten conocer nuevas estructuras de datos y el funcionamiento de un TAD (Tipo Abstracto de Datos).

También se ha llevado a cabo un proceso de mejora del código, modularizando la interfaz gráfica con el usuario.

El resultado es una herramienta completa, útil y fácilmente ampliable.

BIBLIOGRAFÍA

1. Advanced Java 2 Platform: how to Program. Autores: H. M. Deitel, P. J. Deitel, S. E. Santry. Publicación: Upper Saddle River (New Jersey): Prentice Hall, cop. 2002. Colección: How to program series.

2. Fundamentos de algoritmia. Autores: G. Brassard, P. Bratley. Traducción de Rafael García-Bermejo. Revisión técnica de Narciso Martí, Ricardo Peña y Luís Joyanes Aguilar. Publicación: Madrid: Prentice Hall, 2002. 1ª edición en español.

3. Diseño de programas: formalismo y abstracción. Autor: Ricardo Peña Marí. Publicación México: Prentice Hall, D. L. 2005

4. Data Structures and Algorithm Analysis in Java. Autor Mark Allen Weiss, Florida International University. Publicación: Addison-Wesley, 1999

5. Estructuras de datos y métodos algorítmicos: Ejercicios resueltos. Autores: Narciso Martí Oliet, Yolanda Ortega Mallén, José Alberto Verdejo López. Publicación: Madrid: Pearson Prentice-Hall, 2003. Colección Prentice Práctica.

PÁGINA DE AUTORIZACIÓN

Se autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Gonzalo Moreno Pereira Eduardo de la Iglesia Ramírez Cristina Rubert Sánchez