**PRÁCTICA 28 | APLICACIÓN ÁRBOLES**

# Introducción

Es interesante saber que las bases de datos utilizan árboles para optimizar las operaciones que se pueden realizar con ellas, esta aplicación no me había pasado por la cabeza. La práctica luce complicada, espero poder cumplir con su desarrollo completo en tiempo y forma.

Lo más probable es que esta sea la última práctica en cuanto árboles se refiere, así que será una buena forma de despedir el tema.

# Desarrollo

Para dejar de estar fuera de contexto tuve que leer varias veces las instrucciones de la práctica, más que nada porque se mencionaban direcciones de memoria para interactuar con el archivo y sus registros.

Comencé un poco a ciegas porque había entendido la idea esencial, pero había unas cosas que estaban aún un tanto inconclusas. En base a lo requerido pues, empecé a realizar el diagrama de clases que ya es acostumbrado ilustrar en los reportes de las últimas prácticas. En esta ocasión el diagrama jugó un papel más importante, porque la práctica en su mayoría consistía en crear nuevas clases y necesitaba ver que la interacción entre ellas luciera correcta y sus métodos fueran los necesarios también. Me surgió la idea también de crear un nuevo paquete para situar la clase que se encargaría de indexar y aplicar los procedimientos al archivo, en efecto lo cree y se llama **basededatos.**

Teniendo el diseño proseguí a codificar lo planteado, como será visto más adelante. Hubo un detalle con una clase que había hecho en una práctica anterior a la que le había puesto el nombre de **Separador**, el código de su método “separar” lo utilicé para obtener de la cadena de texto de un registro sus atributos individuales, pero trasladé una copia de esta misma función a una nueva clase de nombre **ManipuladorString**, el motivo de todo esto fue el nombre de la primera versión, que no se prestaba mucho a agregarle métodos que también aplicaran efectos a una cadena de texto que no fuesen necesariamente de separación.

Me concentré en codificar primero todas las dependencias de la clase anfitriona de esta práctica, que recibió el nombre de **DAOEmpleadoArchivo**, para así tener la comodidad de implementar la funcionalidad de sus métodos sin necesidad de estar creando más en otras clases para que funcionara como debía.

Fue hasta el método que hace la labor de indexar los registros del archivo en el árbol binario de búsqueda y usando la clase **RandomAccessFile,** contenida en el código anexado para la lectura y manipulación de archivos, que terminé de comprender por completo la práctica.

El desarrollo de los métodos de la clase anfitriona fue variando bastante de complejidad y tiempo que me tomó en hacer.

Comencé por hacer la función de búsqueda de un registro, porque se veía como la más sencilla de todas, y en efecto lo fue. El funcionamiento de éste incluso estaba descrito en la práctica, y usar el método de separación de una cadena de texto, que ya había hecho antes, me agilizó bastante las cosas.

El siguiente que codifiqué fue el método encargado de la inserción de un nuevo registro, también estuvo fácil y rápido de hacer, tal vez lo único que me detuvo un poco fue que el puntero del archivo debía estar una posición adelante de la última que se quedó, de lo contrario, sobrescribía un carácter del registro anterior, cosa que pude solucionar imprimiendo las posiciones de memoria y longitud de cada registro para darme cuenta de lo que estaba haciendo mal.

La situación se demoró y complicó más cuando llegué al procedimiento de eliminar; para este método sí opté con más razón por realizar un análisis previo a su codificación. En mi análisis contemplé el siguiente proceso:

1. Buscar el id del registro en el árbol para obtener su posición en memoria.
2. Mover el puntero del archivo hasta la posición obtenida.
3. A partir de ahí, recorrer todos los registros por debajo del que sería eliminado; sobrescribiendo la información que estaba en la siguiente línea sobre el registro en curso.
4. Reducir el tamaño del archivo en función de la longitud que tenía el registro eliminado.

En sí el método para mí tenía sentido, pero nuevamente a la hora de llevarlo a la práctica surgieron varios errores relacionados a la posición del puntero y duplicación de registros en algunos casos, este segundo, la duplicación, fue uno de los que realmente no tenía una solución muy complicada, pero me tomó un tiempo darme cuenta. El problema radicaba en que, si la línea actual era más pequeña en longitud a la siguiente, era cuando sucedía la duplicidad: esto pasaba porque se necesitaba hacer otro *readline()* al archivo para poder alcanzar el registro que debía ser recorrido en la siguiente iteración. Después de arreglar ese error, el programa lucía funcionando como debería de ser.

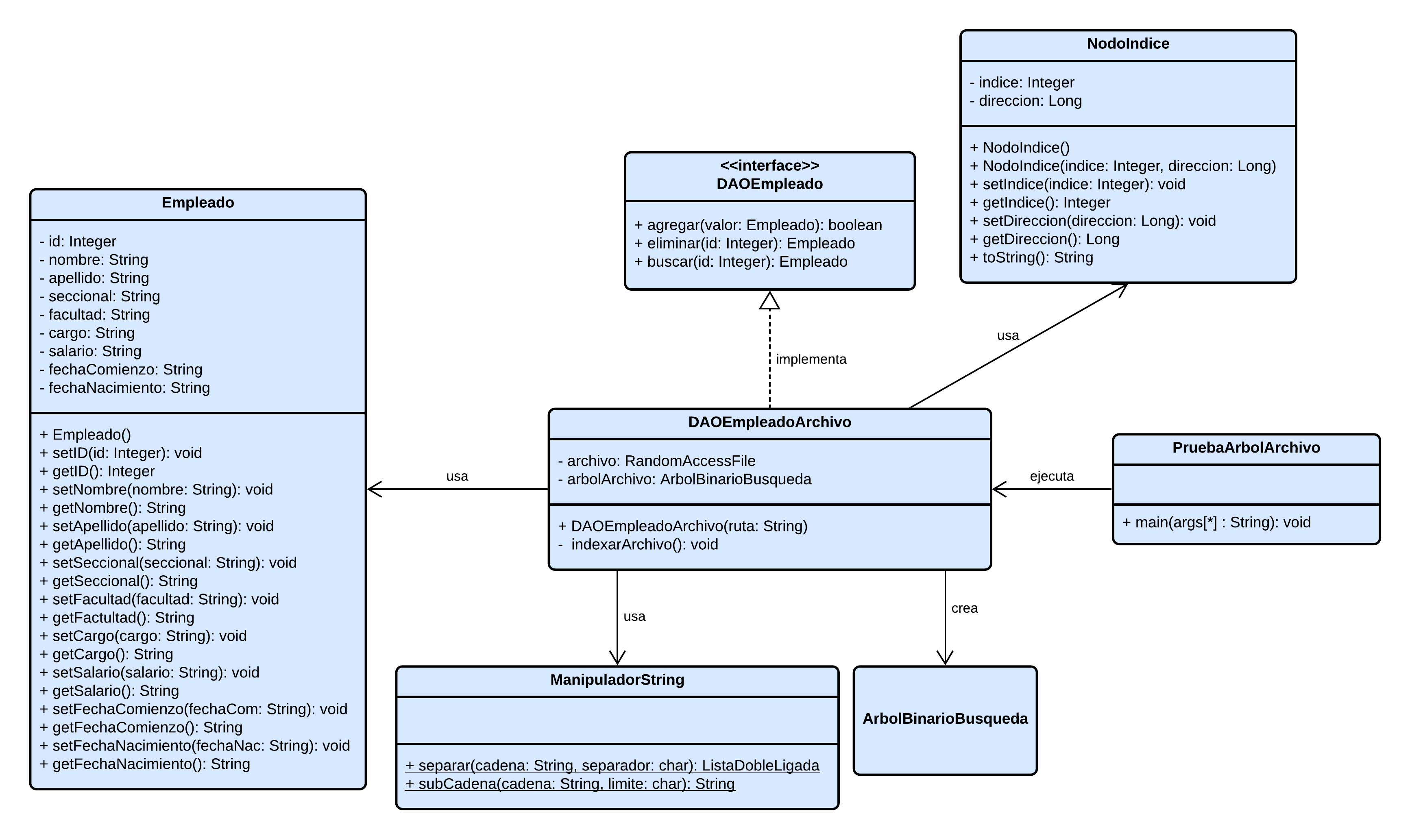
A continuación, se indican los archivos creados y su ubicación dentro del proyecto:

* ***Empleado.java (paquete catalogos).***
* ***NodoIndice.java (paquete catalogos).***
* ***ManipuladorString.java (paquete utilidades).***
* ***DAOEmpleado.java (paquete basededatos).***
* ***DAOEmpleadoArchivo.java (paquete basededatos).***
* ***PruebaArbolArchivo.java (paquete interfaces).***

La clase anterior contiene las pruebas de esta práctica.

# Diagrama de clases

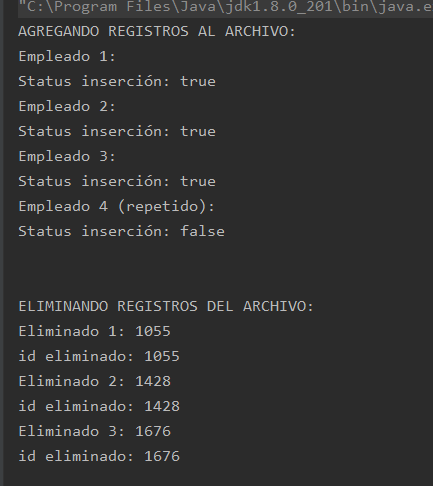
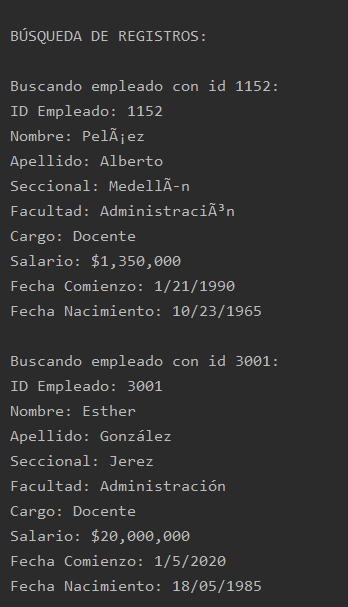
En el diagrama de clases se tiene una vista más cómoda de lo que se hizo en cada una de las clases mencionadas anteriormente. Cabe destacar que para las que ya existían en el proyecto solamente se ilustran los métodos o atributos que fueron agregados. En cambio, para las nuevas se muestra en su totalidad métodos y atributos que fueron incluidos.

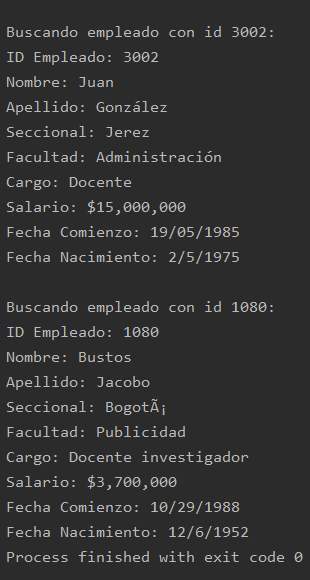


**Si desea revisar la documentación de lo que fue agregado al proyecto vaya a las siguientes rutas:**

* **/docs/catalogos/Empleado.html**
* **/docs/catalogos/NodoIndice.html**
* **/docs/utilidades/ManipuladorString.html**
* **/docs/basededatos/DAOEmpleado.html**
* **/docs/basededatos/DAOEmpleadoArchivo.html**
* **/docs/interfaces/PruebaArbolArchivo.html**

# Capturas del programa funcionando



# Conclusiones

Al igual que he dicho en otras prácticas, nuevamente he aprendido algo muy interesante. Este concepto para manipular las bases de datos es una idea muy buena para optimizar su funcionamiento en gran medida, y también, es otra prueba de que las estructuras de datos tienen muchísimas aplicaciones y presencia en lugares que al menos yo, no imaginaba.