Algoritmos y Estructuras de Datos

Guía de laboratorio 5
Estructuras enlazadas utilizando handles

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

El siguiente taller consiste en la implementación y uso de la clase `Handle`, con la finalidad de obtener mejor eficiencia en ciertas operaciones sobre estructuras de datos enlazadas.

Link de entrega: Formulario Campus Virtual (**Adjuntar** los archivos `ABB.java`, `ListaEnlazada.java` y `SistemaPedidos.java`)

Última fecha de entrega: domingo 09/11 a las 23:59hs.

Consigna

En este taller, modificaremos la implementación de la clase `ABB` para que utilice *handles*. Un *handle* es un objeto que actúa como referencia a otro objeto, permitiendo acceder y modificar el objeto referenciado de manera indirecta en $O(1)$. En este caso, el handle será un objeto que contiene un puntero al nodo de la lista enlazada. De esta manera, podremos acceder y modificar los elementos de la lista sin necesidad de recorrerla.

Luego, implementaremos una clase que combina una lista enlazada con un árbol binario de búsqueda para almacenar pedidos de comida rápida, permitiendo acceder a los pedidos tanto por orden de llegada (usando la lista enlazada) como por su identificador (usando el árbol binario de búsqueda). Se puede asumir que los identificadores siguen una distribución aleatoria, por lo que el árbol binario de búsqueda se mantendrá balanceado en promedio.

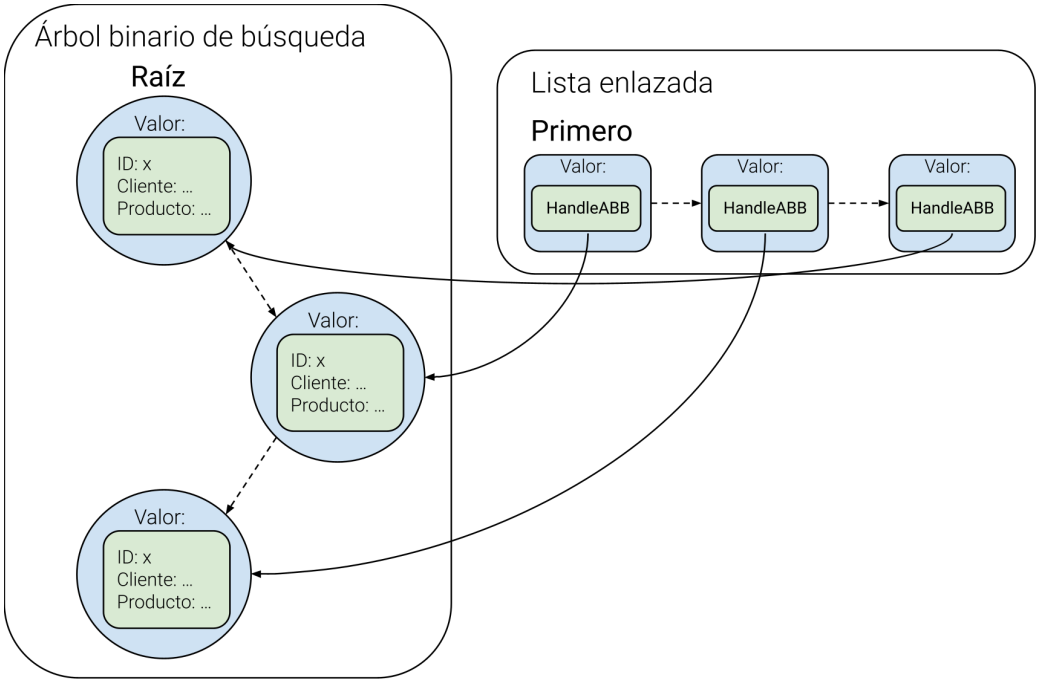

Figura 1: Diagrama de estructuras de datos enlazadas utilizando handles.

Para resolver el taller, deben modificar la clase `ABB<T>` (dentro del archivo `ABB.java`) de manera tal que devuelva un *handle* al agregar un elemento y permitia eliminar elementos a través de él:

1. Definir la clase `HandleABB` que implemente la interfaz `Handle`. Esta clase debe contener una referencia al nodo del árbol binario de búsqueda que contiene el elemento referenciado por el handle.
2. Implementar el método `T valor()` de la clase `HandleABB` para obtener el valor del nodo referenciado por el handle.
3. Implementar el método `void eliminar()` de la clase `HandleABB` para eliminar el nodo referenciado por el handle.

4. Modificar el método `void insertar(T elem)` para que devuelva un *handle* al agregar un elemento al árbol: `HandleABB insertar(T elem)`.

Implementar la clase `SistemaPedidos` (en el archivo `SistemaPedidos.java`) de manera tal que permita agregar pedidos, obtener el pedido con menor identificador, quitar pedidos por orden de llegada y listarlos tanto por su orden de llegada como por su identificador. Recordar que los identificadores siguen una distribución aleatoria, por lo que la complejidad promedio de las operaciones sobre el árbol binario de búsqueda es $O(logn)$.

1. `void agregarPedido(Pedido pedido);`
   Agrega un pedido al sistema con complejidad $O(log\ n)$ en promedio.
2. `Pedido proximoPedido();`
   Devuelve el próximo pedido por orden de llegada y lo elimina del sistema con complejidad $O(log\ n)$.
3. `Pedido pedidoMenorId();`
   Devuelve el pedido con menor identificador con complejidad $O(log\ n)$ en promedio.
4. `String obtenerPedidosEnOrdenDeLlegada();`
   Devuelve un string con los pedidos en orden de llegada en $O(n)$ (e.g., "[28, 71, 17, 261, 21]").
5. `String obtenerPedidosOrdenadosPorId();`
   Retorna un string con los pedidos ordenados por su identificador en $O(n)$ (e.g., "{17, 21, 28, 71, 261}").