



Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Aragón

Ingeniería en Computación

**Asignatura:** Estructura de datos

**Proyecto Final:** Sistema de Gestión de Inventario de Productos

**Profesor:** Jesús Hernández Cabrera

**Alumno:** Juan Diego Ortiz Cruz

**Grupo:** 1360

**Fecha:** 21/11/2024

# Sistema de Gestión de Inventario con Árbol Binario de Búsqueda

## Introducción

### Descripción de la Temática

El presente proyecto desarrolla un sistema de gestión de inventario para una tienda de productos variados, implementando una estructura de datos de Árbol Binario de Búsqueda (ABB) para la organización y administración eficiente del inventario, con una UI WEB moderna y sencilla para su fácil acceso e interacción.

### Objetivos

- Implementar un Árbol Binario de Búsqueda para almacenar productos
- Crear un sistema que permita gestionar el inventario mediante operaciones básicas
- Aplicar conceptos de estructuras de datos y programación genérica en Java

## Estructura de Datos: Árbol Binario de Búsqueda (ABB)

### Características del ABB

- Permite una búsqueda, inserción y eliminación eficiente de elementos
- Mantiene los elementos ordenados según una clave (en este caso, el ID del producto)
- Complejidad temporal promedio  $O(\log n)$  para operaciones principales

### Propiedades

- Cada nodo tiene como máximo dos hijos (izquierdo y derecho)
- Subárbol izquierdo contiene elementos menores a la raíz
- Subárbol derecho contiene elementos mayores a la raíz

# Estructura del Proyecto

## Clases Principales

### 1. Producto

- Clase genérica que representa un producto en el inventario
- Atributos:
  - ID (clave de búsqueda)
  - Nombre
  - Precio
  - Cantidad

```
public class Producto implements Comparable<Producto> { 15 usages
    public String name; 2 usages
    public int idP; 5 usages
    public double precio; 2 usages
    public int cantidad; 2 usages

    // Constructor
    public Producto(String name, int idP, double precio, int cantidad) {
        this.name = name;
        this.idP = idP;
        this.precio = precio;
        this.cantidad = cantidad;
    }

    public int getIdP() { return idP; }

    @Override
    public int compareTo(Producto o) {
        // Comparamos los códigos de producto (idP)
        return Integer.compare(this.idP, o.idP);
    }

    @Override
    public String toString() {
        return String.format("%-4d | %-20s | $%-10.2f | %d",
            idP, name, precio, cantidad);
    }
}
```

## 2. ArbolBinarioBusqueda (ABB)

- Implementa las operaciones del Árbol Binario de Búsqueda
- Integra una clase Nodo privada

```
public class ABB<T extends Comparable<T>> {  
    private class Nodo { 16 usages  
        T dato; 14 usages  
        Nodo izquierdo; 15 usages  
        Nodo derecho; 15 usages  
  
        public Nodo(T dato) { 1 usage  
            this.dato = dato;  
            this.izquierdo = null;  
            this.derecho = null;  
        }  
    }  
}  
  
private Nodo raiz; 10 usages  
  
public ABB() { this.raiz = null; }
```

- Métodos principales:
  - insertar() -> insertarRec()
  - eliminar() -> eliminarRec()
  - buscar() -> buscarRec()
  - encontrarMinimo()
  - InOrden() -> inOrdenRec()
  - inOrdenToList() -> inOrdenToListRec()
  - preOrden() -> preOrdenRec()
  - postOrden() -> postOrdenRec()

# Funcionalidades del Sistema

## Implementación de Operaciones

```
@Service 2 usages
public class InventarioService {
    private final ABB<Producto> inventario; 5 usages
    private final List<Producto> listaProductos;

    public InventarioService() { no usages
        this.inventario = new ABB<>();
        this.listaProductos = new ArrayList<>();
    }
}
```

- **Agregar Producto:** Inserta un nuevo producto en el ABB

```
public void agregarProducto(Producto producto) {
    inventario.insertar(producto);
    actualizarLista();
}
```

- **Eliminar Producto:** Elimina un producto por su ID

```
public void eliminarProducto(int id) { 1 usage
    Producto producto = new Producto( name: "", id, precio: 0, cantidad: 0);
    inventario.eliminar(producto);
    actualizarLista();
}
```

- **Buscar Producto:** Localiza un producto por ID

```
public Producto buscarProducto(int id) { 2 usages
    return inventario.buscar(new Producto( name: "", id, precio: 0, cantidad: 0));
}
```

- **Listar Inventario:** Muestra todos los productos ordenados

```
public List<Producto> obtenerTodosLosProductos() {
    actualizarLista();
    return listaProductos;
}
```

# Implementación de Operaciones

## Inserción

- Complejidad temporal:  $O(\log n)$
- Proceso:
  1. Comparar el ID del nuevo producto con el nodo actual
  2. Moverse al subárbol izquierdo si es menor
  3. Moverse al subárbol derecho si es mayor
  4. Insertar cuando se encuentra una posición vacía

```
public void insertar(T elemento) { raiz = insertarRec(raiz, elemento); }

private Nodo insertarRec(Nodo nodo, T elemento) { 3 usages
    if (nodo == null) {
        return new Nodo(elemento);
    }

    int comparacion = elemento.compareTo(nodo.dato);
    if (comparacion < 0) {
        nodo.izquierdo = insertarRec(nodo.izquierdo, elemento);
    } else if (comparacion > 0) {
        nodo.derecho = insertarRec(nodo.derecho, elemento);
    }

    return nodo;
}
```

## Eliminación

- Maneja dos casos:
  1. Nodo hoja (sin hijos) o con un solo hijo
  2. Nodo con dos hijos (reemplazar con sucesor inorden)

```
private Nodo eliminarRec(Nodo nodo, T elemento) { 4 usages
    if (nodo == null) {
        return null;
    }

    int comparacion = elemento.compareTo(nodo.dato);
    if (comparacion < 0) {
        nodo.izquierdo = eliminarRec(nodo.izquierdo, elemento);
    } else if (comparacion > 0) {
        nodo.derecho = eliminarRec(nodo.derecho, elemento);
    } else {
        // Caso 1: Nodo hoja o con un solo hijo
        if (nodo.izquierdo == null) {
            return nodo.derecho;
        } else if (nodo.derecho == null) {
            return nodo.izquierdo;
        }

        // Caso 2: Nodo con dos hijos
        // Encontrar el sucesor inorden (mínimo valor en subárbol derecho)
        nodo.dato = encontrarMinimo(nodo.derecho);
        // Eliminar el sucesor inorden
        nodo.derecho = eliminarRec(nodo.derecho, nodo.dato);
    }

    return nodo;
}
```

## Búsqueda

- Implementa búsqueda recursiva
- Complejidad temporal:  $O(\log n)$

```

public T buscar(T elemento) { 1 usage
    Nodo resultado = buscarRec(raiz, elemento);
    return resultado == null ? null : resultado.dato;
}

private Nodo buscarRec(Nodo nodo, T elemento) { 3 usages
    if (nodo == null || elemento.compareTo(nodo.dato) == 0) {
        return nodo;
    }

    if (elemento.compareTo(nodo.dato) < 0) {
        return buscarRec(nodo.izquierdo, elemento);
    }

    return buscarRec(nodo.derecho, elemento);
}

```

## Consideraciones Técnicas

### Tecnologías

- Lenguaje: Java
- Programación Genérica
- Estructuras de Datos: Árbol Binario de Búsqueda
- Uso de FrameWorks (SpringBoot)
- Programacion Web para la UI

### UI

- Se desarrollo una UI con Thymeleaf y Tailwind CSS
- clase InventarioController para poder interactuar con los elementos del DOM
- clase InventarioService para manejar todas las operaciones del ABB



## Proyecto en funcionamiento

### Página principal (inventario)





- Llama a `inventarioService.obtenerTodosLosProductos()` para listar todos los productos almacenados
- Botón Nuevo Producto lleva a un formulario para agregar un nuevo producto
- Botón Buscar Producto lleva a la página de buscar
- Botón de Eliminar, este llama a la función eliminar, lo hace por ID

### Inventario de Productos

Producto guardado exitosamente

+ Nuevo Producto

🔍 Buscar Producto

| ID  | NOMBRE                   | PRECIO     | CANTIDAD | ACCIONES   |
|-----|--------------------------|------------|----------|--|
| 114 | Xiaomi 14 Ultra 5G       | \$26999.99 | 2        |  Eliminar   |
| 115 | iPhone 15 Pro            | \$22499.99 | 5        |  Eliminar  |
| 116 | iPhone 16 Pro Max        | \$28999.99 | 4        |  Eliminar |
| 224 | Samsung Galaxy S24 Ultra | \$24999.99 | 6        |  Eliminar |

### Página Nuevo Producto (agregar)

- Cuatro inputs uno para cada atributo del objeto Producto
- Botón Guardar primero busca si el ID introducido ya existe (`inventarioService.buscarProducto(producto.getIdP() != null)`) esto regresara un mensaje de error y no dejara agregar el Producto, si el ID es nuevo entonces se llamara al método `inventarioService.agregarProducto(producto)`
- botón Cancelar regresara a la página anterior (inventario) sin mandar nada

## Nuevo Producto

Nombre:

ID:

Precio:

Cantidad:

Guardar

Cancelar

## Nuevo Producto

Ya existe un producto con ese ID

Nombre:

ID:

Precio:

Cantidad:

Guardar

Cancelar

## Página Buscar Producto (buscar)

- Elemento Input para introducir el ID del producto a buscar
- Botón Buscar llama a el método inventarioService.buscarProducto(id)
- Botón Volver al Inventario este nos regresa a la página anterior (inventario)

### Buscar Producto

BuscarVolver al Inventario

### Buscar Producto

Buscar

No se encontró el producto con ID: 100

Volver al Inventario

### Buscar Producto

Buscar

**iPhone 16 Pro Max**

**ID:** 116

**Precio:** \$28999.99

**Cantidad:** 4

Volver al Inventario