UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGON

PROYECTO FINAL

P R E S E N T A

Alexis Hernández Zamudio

APROFESOR

Jesús Hernández Cabrera

Gpo:1158

URL del repositorio:

https://github.com/TyrBalder1439/Estructura-de-Datos-

Ciudad Nezahualcóyotl, EDOMEX. 19 DE NOVIEMBRE DEL 2024

LA TEMATICA UTILIZADA EN ESTE PROYECTO FUE DE LA VENTA DE TACOS.

----------Clase Producto----------

GestorDeInventarios.java | Producto.java | ArbolBinarioBusqueda.java

```java
public class Producto implements Comparable<Producto> {  14 usages
    private String nombre;  4 usages
    private int id;  6 usages
    private double precio;  4 usages
    private int cantidad;  4 usages
    public Producto(String nombre, int id, double precio, int cantidad) {  3 u
        this.nombre = nombre;
        this.id = id;
        this.precio = precio;
        this.cantidad = cantidad;
    }
    public void setCantidad(int cantidad) {  no usages
        this.cantidad = cantidad;
    }
    public void setPrecio(double precio) {  no usages
        this.precio = precio;
    }
    public void setId(int id) {  no usages
        this.id = id;
    }
    public void setNombre(String nombre) {  no usages
        this.nombre = nombre;
    }

    public String getNombre() {  no usages
        return nombre;
    }
    public int getId() {  no usages
        return id;
    }
    public double getPrecio() {  no usages
        return precio;
    }
    public int getCantidad() {  no usages
        return cantidad;
    }
    @Override
    public int compareTo(Producto otro) {
        return Integer.compare(this.id, otro.id);
    }
    @Override
    public String toString() {
        return "Producto{" +
                "ID= " + id +
                ", Nombre = '" + nombre + '\'' +
                ", Precio = $" + precio +
                ", Cantidad= " + cantidad +
                '}';
    }
}
```

CLASE PRODUCTO

La clase Producto es una implementación de la interfaz Comparable<Producto>, por lo que permite saber si los objetos de esta clase son capaces de ser comparados entre sí, con lo que es posible ordenarlos.

Atributos: La clase contiene cuatro atributos privados:

- nombre: un String que almacena el nombre del producto.
- id: un int que indica un identificador único para el producto.
- precio: un double que indica el precio del producto.
- cantidad: un int que indica cuántas unidades del producto se encuentran disponibles.

Constructor: El constructor de la clase permite instanciar un objeto Producto inicializando las propiedades de la forma indicada cuando se crea el objeto.

Métodos set y get: Se incluyen métodos para fijar (set) y conseguir (getter) los valores de los atributos. Con esto se consigue encapsular los datos y controlar los accesos a los mismos.

Método compareTo: Este método permite comparar dos objetos Producto mediante el atributo id. El método devuelve un valor negativo, cero o positivo dependiendo de si el id del objeto actual es menor, igual o mayor que el id del objeto pasado como argumento.

Método toString: El método sobrescribe el método toString de la clase Object y proporciona una representación en forma de cadena del objeto Producto mostrando los atributos de forma legible.

En resumen, esta clase es una representación simple de un producto que incluye atributos, métodos para los atributos y la posibilidad de compararse con otros productos. Esto puede ser útil en aplicaciones que gestionan inventarios o catálogos de productos.

----------Clase ArbolBinarioBusqueda----------

GestorDeInventarios.java | Producto.java | ArbolBinarioBusqueda.java

```java
public class ArbolBinarioBusqueda<T extends Comparable<T>> {  6 usages
    private class Nodo {  11 usages
        T dato;  13 usages
        Nodo izquierdo;  11 usages
        Nodo derecho;  12 usages

        Nodo(T dato) {  1 usage
            this.dato = dato;
            izquierdo = derecho = null;
        }
    }

    private Nodo raiz;  6 usages

    public void insertar(T elemento) {  1 usage
        raiz = insertarRec(raiz, elemento);
    }

    private Nodo insertarRec(Nodo nodo, T elemento) {  3 usages
        if (nodo == null) {
            return new Nodo(elemento);
        }
        if (elemento.compareTo(nodo.dato) < 0) {
            nodo.izquierdo = insertarRec(nodo.izquierdo, elemento);
        } else if (elemento.compareTo(nodo.dato) > 0) {
            nodo.derecho = insertarRec(nodo.derecho, elemento);
        } else {
            throw new IllegalArgumentException("El elemento ya existe en el árbol");
        }
        return nodo;
    }

    public void eliminar(T elemento) {  1 usage
        raiz = eliminarRec(raiz, elemento);
    }
```

```java
36
37      private Nodo eliminarRec(Nodo nodo, T elemento) {  4 usages
38          if (nodo == null) {
39              throw new IllegalArgumentException("El elemento no existe en el árbol");
40          }
41          if (elemento.compareTo(nodo.dato) < 0) {
42              nodo.izquierdo = eliminarRec(nodo.izquierdo, elemento);
43          } else if (elemento.compareTo(nodo.dato) > 0) {
44              nodo.derecho = eliminarRec(nodo.derecho, elemento);
45          } else {
46              if (nodo.izquierdo == null) {
47                  return nodo.derecho;
48              } else if (nodo.derecho == null) {
49                  return nodo.izquierdo;
50              }
51              nodo.dato = obtenerMinimo(nodo.derecho);
52              nodo.derecho = eliminarRec(nodo.derecho, nodo.dato);
53          }
54          return nodo;
55      }
56
57      private T obtenerMinimo(Nodo nodo) {  1 usage
58          T minimo = nodo.dato;
59          while (nodo.izquierdo != null) {
60              nodo = nodo.izquierdo;
61              minimo = nodo.dato;
62          }
63          return minimo;
64      }
65
66      public T buscar(T elemento) {  1 usage
67          return buscarRec(raiz, elemento);
68      }
```

```java
70      private T buscarRec(Nodo nodo, T elemento) {  3 usages
71          if (nodo == null) {
72              return null;
73          }
74          if (elemento.compareTo(nodo.dato) < 0) {
75              return buscarRec(nodo.izquierdo, elemento);
76          } else if (elemento.compareTo(nodo.dato) > 0) {
77              return buscarRec(nodo.derecho, elemento);
78          } else {
79              return nodo.dato;
80          }
81      }
82
83      public void inOrden() {  1 usage
84          inOrdenRec(raiz);
85      }
86
87      private void inOrdenRec(Nodo nodo) {  3 usages
88          if (nodo != null) {
89              inOrdenRec(nodo.izquierdo);
90              System.out.println(nodo.dato+" ");
91              inOrdenRec(nodo.derecho);
92          }
93      }
94  }
95
```

CLASE ARBOL BINARIO DE BUSQUEDA (ABB)

La clase ArbolBinarioBusqueda es una clase propia que tiene un tipo genérico T que implementa la interfaz Comparable, lo que significa que aquellos elementos que se almacenan dentro del árbol deben ser comparables entre si.

La clase ArbolBinarioBusqueda contiene un nodo interno que es de tipo Nodo y que representa un nodo del árbol. Cada uno de los nodos tiene un valor dato y, además, tiene tanto un hijo izquierdo como un hijo derecho.

La clase ArbolBinarioBusqueda tiene varios métodos, que son:

- insertar(T elemento): que inserta en el árbol un nuevo elemento. Si el elemento ya existe, lanza una excepción("El elemento ya existe en el árbol");
- eliminar(T elemento): que elimina del árbol un elemento. Si el elemento no existe, lanza una excepción ("El elemento no existe en el árbol");
- buscar(T elemento): que busca en el árbol un elemento y devuelve el valor si existe, o devuelve null si no existe;
- inOrden(): que imprime los elementos del árbol en orden de elementos inorden (izquierdo, raíz y derecho).

El método insertarRec es un método de carácter recursivo que permite insertar un nuevo elemento dentro del árbol, si el nodo es null crea un nuevo nodo cuyo valor será el de este elemento. En el caso que el elemento sea menor que el valor del nodo actual ejecuta insertarRec al hijo izquierdo. Por el contrario en el caso de que el elemento sea mayor que el valor del nodo actual ejecuta insertarRec al hijo derecho.

El método eliminarRec es un método de carácter recursivo que permite eliminar un elemento del árbol en caso de que el nodo en cuestión sea null se lanza una excepción, en caso de que el elemento sea menor que el del nodo actual se ejecuta eliminarRec al hijo izquierdo, si el elemento es mayor que el valor del nodo actual ejecuta eliminarRec a su hijo derecho, y en el caso en que el elemento es igual que el del nodo actual se elimina el nodo y se sustituye su valor por el mínimo valor del subárbol derecho.

El método obtenerMinimo devuelve el mínimo valor de un subárbol.

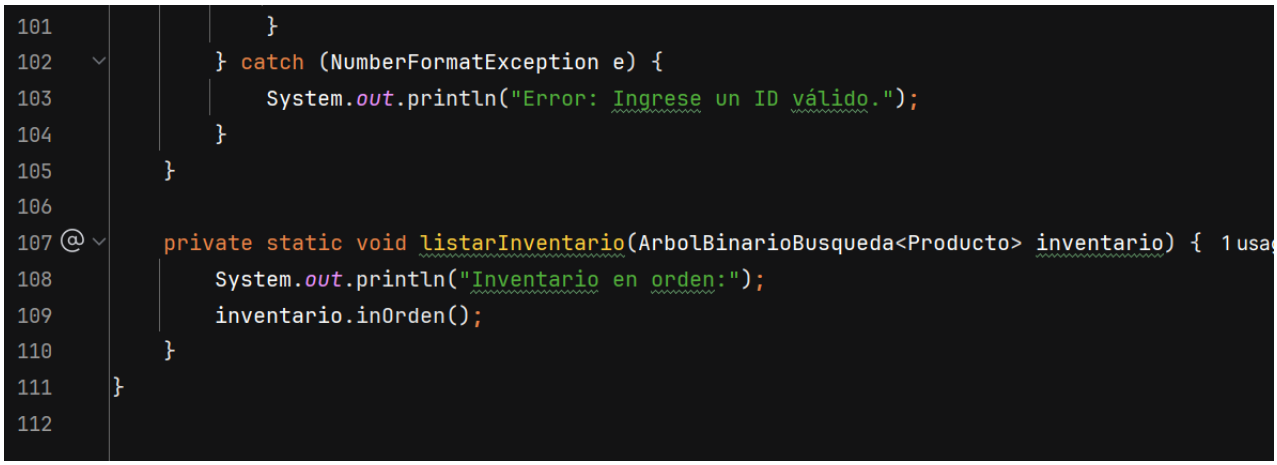
El método buscarRec es un método recursivo que busca un elemento en el árbol. Si el nodo es null devolverá null, si el elemento es menor que el del nodo actual ejecuta buscarRec al hijo izquierdo si el elemento es mayor que el del nodo actual ejecuta buscarRec al hijo derecho, si el elemento es igual que el dato del nodo devuelvo su valor.

El método inOrdenRec: Este método recursivo realiza el recorrido en orden. Si el nodo no es null, primero llama a sí mismo para el subárbol izquierdo, luego imprime el dato del nodo actual y finalmente llama a sí mismo para el subárbol derecho. Esto resulta en una salida ordenada de los elementos del árbol.

----------CLASE GESTION DE INVENTARIO----------

```java
import java.util.Scanner;

public class GestorDeInventarios {
    public static void main(String[] args) {
        ArbolBinarioBusqueda<Producto> inventario = new ArbolBinarioBusqueda<>();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            try {
                System.out.println("\n--- Gestión de Inventario ---");
                System.out.println("1. Agregar Producto");
                System.out.println("2. Eliminar Producto");
                System.out.println("3. Buscar Producto");
                System.out.println("4. Listar Inventario");
                System.out.println("5. Salir");
                System.out.print("Seleccione una opción: ");
                int opcion = Integer.parseInt(scanner.nextLine());

                switch (opcion) {
                    case 1:
                        agregarProducto(scanner, inventario);
                        break;
                    case 2:
                        eliminarProducto(scanner, inventario);
                        break;
                    case 3:
                        buscarProducto(scanner, inventario);
                        break;
                    case 4:
                        listarInventario(inventario);
                        break;
                    case 5:
                        System.out.println("Saliendo del sistema...");
                        return;
                    default:
                        System.out.println("Opción inválida. Intente de nuevo.");
                }
            } catch (NumberFormatException e) {
                System.out.println("Entrada inválida. Por favor, ingrese un número válido.");
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
    }

    private static void agregarProducto(Scanner scanner, ArbolBinarioBusqueda<Producto> inventario) {
        try {
            System.out.print("Nombre del producto: ");
            String nombre = scanner.nextLine();

            System.out.print("ID del producto: ");
            int id = Integer.parseInt(scanner.nextLine());
```

```
            System.out.print("Precio del producto: ");
            double precio = Double.parseDouble(scanner.nextLine());

            System.out.print("Cantidad en inventario: ");
            int cantidad = Integer.parseInt(scanner.nextLine());

            if (precio < 0 || cantidad < 0) {
                System.out.println("El precio y la cantidad deben ser positivos.");
                return;
            }

            Producto producto = new Producto(nombre, id, precio, cantidad);
            inventario.insertar(producto);
            System.out.println("Producto agregado con éxito.");
        } catch (NumberFormatException e) {
            System.out.println("Error: Ingrese valores válidos para ID, precio y cantidad.");
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private static void eliminarProducto(Scanner scanner, ArbolBinarioBusqueda<Producto> inventario) {
        try {
            System.out.print("ID del producto a eliminar: ");
            int idEliminar = Integer.parseInt(scanner.nextLine());

            Producto productoEliminar = new Producto("", idEliminar, 0, 0);
            inventario.eliminar(productoEliminar);
            System.out.println("Producto eliminado con éxito.");
        } catch (NumberFormatException e) {
            System.out.println("Error: Ingrese un ID válido.");
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private static void buscarProducto(Scanner scanner, ArbolBinarioBusqueda<Producto> inventario) {
        try {
            System.out.print("ID del producto a buscar: ");
            int idBuscar = Integer.parseInt(scanner.nextLine());

            Producto productoBuscar = new Producto("", idBuscar, 0, 0);
            Producto encontrado = inventario.buscar(productoBuscar);
            if (encontrado != null) {
                System.out.println("Producto encontrado: " + encontrado);
            } else {
                System.out.println("Producto no encontrado.");
            }
```



```
101             }
102         } catch (NumberFormatException e) {
103             System.out.println("Error: Ingrese un ID válido.");
104         }
105     }
106
107 @   private static void listarInventario(ArbolBinarioBusqueda<Producto> inventario) {  1 usa
108         System.out.println("Inventario en orden:");
109         inventario.inOrden();
110     }
111 }
112
```

GESTOR DE INVENTARIO

Importaciones y Clase Principal:

Importamos java.util.Scanner para poder recibir datos por teclado

La clase GestorDeInventarios contiene el método main que es donde inicia el programa.

Estructura del Menú:

Creamos un bucle while(true) para poder estar interactuando de forma continua hasta que el usuario se decida por salir.Dentro de ese bucle se le presenta al usuario un menú con opciones para añadir, eliminar, buscar y listar productos así como para salir del programa.

Manejo de Opciones: Se utiliza un switch para gestionar la opción seleccionada por el usuario. Según la opción elegida, se invocan diferentes métodos (agregarProducto, eliminarProducto, buscarProducto, listarInventario).

Método agregarProducto: Este método permite al usuario ingresar los detalles de un nuevo producto (nombre, ID, precio y cantidad). Se verifica que el precio y la cantidad sean positivos antes de crear un objeto Producto y añadirlo al inventario.

Método eliminarProducto: Permite al usuario eliminar un producto del inventario ingresando su ID. Se crea un objeto Producto con el ID proporcionado y se llama al método eliminar del árbol binario de búsqueda.

Método buscarProducto: Facilita la búsqueda de un producto en el inventario por su ID. Si se encuentra el producto, se muestra su información; de lo contrario, se notifica que no se encontró.

Método listarInventario: Este método imprime todos los productos en el inventario en orden, utilizando el método inOrden del árbol binario de búsqueda.

Manejo de Errores: Se implementan bloques try-catch para gestionar excepciones, como entradas no válidas, asegurando que el programa no se detenga ante errores.

EJECUCION DEL PROGRAMA

- AGREGAR PRODUCTO

```
--- Gestión de Inventario ---
1. Agregar Producto
2. Eliminar Producto
3. Buscar Producto
4. Listar Inventario
5. Salir
Seleccione una opción: 1
Nombre del producto: Toco al pastor
ID del producto: 2
Precio del producto: 15
Cantidad en inventario: 200
Producto agregado con éxito.
```

- ELIMINAR PRODUCTO

```
--- Gestión de Inventario ---
1. Agregar Producto
2. Eliminar Producto
3. Buscar Producto
4. Listar Inventario
5. Salir
Seleccione una opción: 2
ID del producto a eliminar: 2
Producto eliminado con éxito.
```

- BUSCAR PRODUCTO

```
--- Gestión de Inventario ---
1. Agregar Producto
2. Eliminar Producto
3. Buscar Producto
4. Listar Inventario
5. Salir
Seleccione una opción: 3
ID del producto a buscar: 6
Producto encontrado: Producto{ID= 6, Nombre = 'TACOS DE SUADERO ', Precio = $20.0, Cantidad= 250}
```

- LISTAR INVENTARIO

```
--- Gestión de Inventario ---
1. Agregar Producto
2. Eliminar Producto
3. Buscar Producto
4. Listar Inventario
5. Salir
Seleccione una opción: 4
Inventario en orden:
Producto{ID= 1, Nombre = 'CAFE DE OLLA', Precio = $20.0, Cantidad= 200}
Producto{ID= 2, Nombre = 'TACO AL PASTOR ', Precio = $20.0, Cantidad= 200}
Producto{ID= 4, Nombre = 'TACO DE TRIPA ', Precio = $20.0, Cantidad= 150}
Producto{ID= 5, Nombre = 'TACO DE CHORIZO ', Precio = $17.0, Cantidad= 200}
Producto{ID= 6, Nombre = 'TACOS DE SUADERO ', Precio = $20.0, Cantidad= 250}
Producto{ID= 20, Nombre = 'TORTA DE TACOS ', Precio = $35.0, Cantidad= 180}
```

- Ejemplo de duplicación de ID

```
--- Gestión de Inventario ---
1. Agregar Producto
2. Eliminar Producto
3. Buscar Producto
4. Listar Inventario
5. Salir
Seleccione una opción: 1
Nombre del producto: tacos de birria
ID del producto: 2
Precio del producto: 12
Cantidad en inventario: 23
Error: El elemento ya existe en el árbol
```

- Eliminación de productos inexistentes.

```
--- Gestión de Inventario ---
1. Agregar Producto
2. Eliminar Producto
3. Buscar Producto
4. Listar Inventario
5. Salir
Seleccione una opción: 2
ID del producto a eliminar: 5
Error: El elemento no existe en el árbol
```

- Intentos de búsqueda de ID inexistentes

```
--- Gestión de Inventario ---
1. Agregar Producto
2. Eliminar Producto
3. Buscar Producto
4. Listar Inventario
5. Salir
Seleccione una opción: 3
ID del producto a buscar: 9
Producto no encontrado.
```

- Salir

```
--- Gestión de Inventario ---
1. Agregar Producto
2. Eliminar Producto
3. Buscar Producto
4. Listar Inventario
5. Salir
Seleccione una opción: 5
Saliendo del sistema... BYE =)
```