PROYECTO FINAL

Este archivo se realiza con el próposito de hablar tanto del proyecto como el ejercicio de poo, explicando detalles técnicos para comprender mejor lo realizado en cada uno, y lo que se implemento en cada uno de las partes.

PROYECTO

1. Arquitectura y Componentes Principales

El sistema ha sido desarrollado en Java, adhiriéndose a una arquitectura modular para una mejor organización y mantenimiento del código. Se compone principalmente de las siguientes clases:

SistemaInventario.java: Esta es la clase central que alberga la lógica principal del programa. Se encarga de presentar el menú de usuario, gestionar las interacciones de entrada y salida, y coordinar todas las operaciones del inventario, desde el registro hasta el despacho de productos. Su función es actuar como el cerebro que orquesta las llamadas a las diferentes estructuras de datos y algoritmos.

Producto.java: Representa la entidad fundamental del inventario. Esta clase encapsula todos los atributos relevantes de un producto, como su código, nombre, cantidad disponible y precio. Para asegurar un acceso y una modificación controlada de estas propiedades, la clase Producto incluye métodos getters y setters. Además, cuenta con un método toString() sobreescrito que proporciona una representación legible y formateada del objeto, lo cual es muy útil para la visualización en consola. La existencia de esta clase es vital para la encapsulación de los datos del inventario.

ArbolBinario.java: Esta clase implementa una estructura de árbol binario de búsqueda. Su propósito específico en este sistema es organizar y permitir la recuperación de los nombres de los productos en estricto orden alfabético. Facilita operaciones de inserción y recorrido que mantienen la propiedad de ordenamiento del árbol.

Nodo.java: Actúa como una clase auxiliar para ArbolBinario. Define la estructura básica de cada nodo dentro del árbol, conteniendo un valor (que en este caso es el nombre de un producto como un String) y referencias a sus nodos hijos izquierdo y derecho.

2. Estructuras de Datos Implementadas en Detalle

La selección estratégica de las estructuras de datos es el pilar de la eficiencia del sistema.

2.1. Map (Implementado como HashMap)

El Map, específicamente el HashMap, es la estructura de datos central que sirve como la fuente principal y autoritativa de todos los datos del inventario. Cada producto se almacena utilizando su código (String) como una clave única, lo que permite una identificación inequívoca. La principal ventaja técnica del HashMap radica en su funcionamiento basado en Tablas Hash. Esto significa que las operaciones de inserción, búsqueda y eliminación de productos por su código se realizan en un tiempo promedio de O(1) (tiempo constante). Esta característica es fundamental para la escalabilidad del sistema, ya que el rendimiento apenas se ve afectado incluso con un volumen considerablemente alto de productos. El HashMap se utiliza extensivamente en todas

las operaciones clave de gestión del inventario, incluyendo el registro, la búsqueda, la modificación y la eliminación de productos, garantizando una respuesta rápida y eficiente.

2.2. ArrayList

El ArrayList (listaProductos) se utiliza para mantener una lista de los objetos Producto en el orden específico en que fueron registrados en el sistema. Esta estructura proporciona la flexibilidad de un array dinámico, permitiendo un acceso rápido a los elementos por su índice (también en tiempo O(1)) y facilitando las iteraciones secuenciales. Su principal caso de uso es en las funcionalidades de listado de productos donde se desea preservar el orden de inserción original, ofreciendo una vista alternativa del inventario a la que proporciona el HashMap.

2.3. Stack (Pila)

La Stack (Pila) es una estructura de datos que implementa la lógica LIFO (Last-In, First-Out), es decir, el último elemento que se añade es el primero en ser retirado. En nuestro sistema, los productos son empujados (push()) a la pila al momento de su registro. Las operaciones de push, pop (sacar el último) y peek (ver el último sin retirarlo) se realizan en tiempo constante (O(1)). Esta pila se utiliza para simular una lista de los "últimos productos ingresados", lo cual podría ser útil en funcionalidades como un historial de acciones recientes o una opción para "deshacer" en sistemas más complejos. Es importante notar que, actualmente, el pop() de la pila también elimina el producto del inventario principal, lo cual es un comportamiento de diseño que podría requerir ajuste si la intención fuera solo un procesamiento temporal sin afectar el stock central.

2.4. Queue (Cola)

La Queue (Cola), implementada usando un LinkedList, es una estructura de datos que adhiere al principio FIFO (First-In, First-Out), lo que significa que el primer elemento que se añade es el primero en ser procesado. Los productos se añaden (offer()) a la cola al ser registrados y se retiran (poll()) para su procesamiento. Al igual que la pila, las operaciones offer y poll son de tiempo constante (O(1)). La cola es fundamental para la funcionalidad de "despacho de productos", donde simula una línea de espera o un flujo de trabajo en el que los ítems se atienden en el orden estricto de su llegada. Similar a la Stack, la operación poll() en este sistema también elimina el producto del inventario principal, un punto a considerar para futuras evoluciones si el despacho solo implica una reducción de cantidad o un movimiento entre estados.

2.5. Vectores (Arrays Tradicionales)

El sistema también utiliza vectores (arrays tradicionales) para almacenar datos de productos de forma paralela, como String[] codigos, String[] nombres, int[] cantidades, y double[] precios. Aunque el acceso a elementos por índice es rápido (O(1)), su principal uso aquí es como estructuras auxiliares para la implementación del Método Burbuja en el ordenamiento. Es relevante mencionar que, en un sistema más maduro o en un diseño más optimizado, la redundancia de datos entre estos arrays y el HashMap (inventario) se eliminaría para simplificar el mantenimiento, reducir el riesgo de inconsistencias y centralizar la información en una única fuente principal.

3. Algoritmos Implementados

Los algoritmos juegan un papel crucial en la manipulación y organización eficiente de los datos dentro del sistema.

3.1. Árbol Binario de Búsqueda (Clase ArbolBinario y Nodo)

La implementación de un Árbol Binario de Búsqueda a través de las clases ArbolBinario y Nodo es una característica destacada. Este algoritmo se basa en una estructura jerárquica donde cada Nodo contiene un valor (el nombre de un producto, como un String) y hasta dos referencias a nodos hijos: uno izquierdo y otro derecho. La propiedad fundamental del árbol binario de búsqueda es que todos los valores en el subárbol izquierdo de un nodo son menores que el valor del nodo padre, y todos los valores en el subárbol derecho son mayores. El método insertar(String nombre) utiliza un enfoque recursivo (insertarRec) para añadir nuevos nombres al árbol, manteniendo esta propiedad de ordenamiento y empleando compareToIgnoreCase() para asegurar una ordenación alfabética que no distingue entre mayúsculas y minúsculas. El método mostrarInOrden() (también recursivo a través de mostrarInOrdenRec) recorre el árbol de una manera específica (izquierda -> raíz -> derecha), garantizando que los nombres de los productos se impriman en estricto orden alfabético. Esto lo hace una herramienta poderosa para listar datos de forma ordenada.

3.2. Método Burbuja (Bubble Sort)

El Método Burbuja (Bubble Sort) es un algoritmo de ordenamiento que hemos implementado manualmente en el sistema. Su funcionamiento es relativamente sencillo: itera repetidamente sobre la lista de elementos, comparando pares de elementos adyacentes y intercambiándolos si no están en el orden deseado. Este proceso se repite hasta que ya no se realizan más intercambios en una pasada completa, lo que indica que la lista está completamente ordenada. En nuestro sistema, se utiliza mediante bucles for anidados para ordenar los arrays paralelos (codigos, nombres, cantidades, precios) basándose en los valores de los precios o los nombres de los productos. Si bien es un algoritmo fácil de entender y de implementar, es importante destacar que su complejidad temporal es de O(n^2). Esto lo hace ineficiente para conjuntos de datos grandes, donde algoritmos más avanzados como QuickSort o MergeSort serían preferibles por su mejor rendimiento. No obstante, cumple su función pedagógica al demostrar un algoritmo de ordenamiento básico.

EJERCICIO POO

1. Clase reserva

Esta clase abstracta define la base para todas las reservas. Sus atributos clave son el nombre del cliente, el número de personas y la cantidad de noches. Lo más importante es el método abstracto costo(), que fuerza a cualquier tipo de reserva a implementar su propia lógica para calcular el precio. Esto asegura que cada reserva sepa cómo calcular su propio costo.

2. Clases ReservaNormal y ReservaPremium

Estas dos clases heredan de Reserva y representan tipos específicos de reservas, mostrando un claro ejemplo de polimorfismo.

-ReservaNormal maneja reservas estándar, añadiendo un valor base por persona y noche (valReservaNormal).

-ReservaPremium está diseñada para reservas con servicios adicionales, incluyendo un valor base (valReservaPremium) y un costoExtra fijo.

Ambas clases implementan de forma polimórfica el método costo(), calculando el precio según sus propias reglas. También personalizan el método mostrarInfo() para presentar sus detalles específicos.

3. Clase Hotel

La clase Hotel gestiona la información de un hotel y su reserva asociada. Su atributo principal es una instancia de Reserva, lo que permite que un mismo objeto Hotel pueda manejar cualquier tipo de reserva (normal o premium) sin importar su tipo exacto. Esto es posible gracias al polimorfismo. Los métodos asignarReserva() y mostrarReserva() demuestran cómo el Hotel interactúa de manera genérica con cualquier Reserva, adaptándose automáticamente a su comportamiento específico.

4. Clase Main (Ejecución del Sistema)

La clase Main es el punto de inicio de la aplicación y demuestra la interacción entre todas las clases. Aquí se crean diferentes tipos de reservas (ReservaNormal y ReservaPremium) y se asignan dinámicamente al Hotel. Se observa cómo el polimorfismo permite que el Hotel trate a todas las reservas de manera uniforme a través de la interfaz Reserva, mientras que el comportamiento real de cálculo de costos o visualización de información se adapta automáticamente en tiempo de ejecución al tipo de reserva específica.