UNIVERSIDAD REGIONAL DE GUATEMALA

CURSO: ALGORITMOS Y PROCESOS II

Proyecto Final

Descripción General:

El objetivo de este proyecto es desarrollar una aplicación en Java que simule las operaciones básicas de un supermercado, utilizando estructuras de datos avanzadas:

- **Árbol AVL** para la gestión eficiente del inventario de productos (búsquedas rápidas y datos siempre ordenados)
- **Cola de Prioridad** para administrar la atención de clientes en las cajas de manera justa y eficiente según su categoría

El sistema debe permitir registrar, buscar, actualizar y eliminar productos del inventario, manteniendo siempre el balance automático del árbol AVL. Asimismo, debe administrar la atención de clientes según su nivel de prioridad (VIP, Adultos Mayores, Clientes Regulares) empleando una cola de prioridad.

Nota: Este proyecto combina dos estructuras de datos fundamentales. Se evalúa tanto la correcta implementación de las estructuras como el funcionamiento integrado del sistema.

REQUERIMIENTOS FUNCIONALES:

1. Módulo de Inventario (Árbol AVL) — 20 pts

1.1 Implementación del Árbol AVL — 10 pts

Implementar un Árbol AVL completo para almacenar los productos del supermercado.

Estructura del Nodo de Producto: Cada nodo del árbol debe contener una clase Producto con los siguientes atributos:

- **Código del producto** (String, ej: "PROD001") — este será la **clave única** para ordenar el árbol
- **Nombre del producto** (String, ej: "Aceite de oliva")
- **Categoría** (String, ej: "Alimentos", "Bebidas", "Electrónica")
- **Precio** (double, ej: 45.99)
- **Cantidad en stock** (int, cantidad disponible en el almacén)
- **Factor de balance** (int) — requerido internamente por el AVL para mantener el árbol equilibrado

Especificaciones técnicas del Árbol AVL:

- El árbol debe auto-equilibrarse tras cada inserción o eliminación
- Implementar rotaciones simples (izquierda/derecha) y rotaciones dobles (izquierda-derecha/derecha-izquierda) cuando sea necesario
- Calcular y actualizar correctamente la altura de cada nodo
- Usar el factor de balance (altura izquierda - altura derecha) para determinar si es necesario rebalancear

1.2 Operaciones del Inventario — 10 pts

Implementar los siguientes métodos en la clase del Árbol AVL. Se debe mostrar claramente en la salida cuándo se realiza cada operación:

- **Insertar producto** (2 pts)
    - Agregar un nuevo producto al árbol
    - Validar que no exista un producto con el mismo código
    - El árbol se rebalanceará automáticamente
    - Mostrar mensajes: "Producto PROD001 insertado correctamente" o "Error: Producto duplicado"
- **Buscar producto por código** (2 pts)
    - Realizar una búsqueda eficiente en el árbol
    - Retornar los datos completos del producto si lo encuentra
    - Mostrar: "Producto encontrado: [detalles]" o "Producto no encontrado"
- **Actualizar datos de un producto** (2 pts)
    - Permitir modificar: nombre, categoría, precio y/o cantidad en stock
    - La búsqueda debe realizarse por código (la clave del árbol no cambia)
    - Mostrar: "Producto actualizado exitosamente" o "Producto no existe"
    - Validar que los nuevos valores sean válidos (precio > 0, cantidad >= 0)
- **Eliminar producto del árbol** (2 pts)
    - Remover un producto por su código
    - El árbol se rebalanceará automáticamente tras la eliminación
    - Mostrar: "Producto eliminado" o "Producto no encontrado"
    - Considerar los tres casos: nodo hoja, nodo con un hijo, nodo con dos hijos

- **Mostrar inventario ordenado (Recorrido In-Orden)** (2 pts)
    - Realizar un recorrido in-orden del árbol (izquierda-nodo-derecha)
    - Mostrar todos los productos en orden alfabético/numérico por código
    - Formato sugerido:
    - === INVENTARIO DEL SUPERMERCADO ===Código | Nombre | Categoría | Precio | Stock-------|------------------|--------------|--------|-------PROD001| Aceite de oliva | Alimentos | $45.99 | 50PROD002| Leche entera | Bebidas | $12.50 | 100

2. Módulo de Atención de Clientes (Cola de Prioridad) — 10 pts

2.1 Implementación de la Cola de Prioridad — 5 pts

Implementar una Cola de Prioridad (puede usar un Min Heap o Max Heap) para gestionar la atención de clientes.

Estructura del Cliente: Cada cliente en la cola debe tener:

- **Nombre** (String, ej: "Juan Pérez")
- **Tipo de cliente** (enum o String: "VIP", "ADULTO_MAYOR", "REGULAR")
- **Número de productos** (int, cantidad de artículos que va a comprar)
- **Hora de llegada** (long o timestamp) — útil para casos de igual prioridad (FIFO)

Definición de Prioridades: Establecer el siguiente orden de atención (de mayor a menor prioridad):

1. **VIP** (prioridad = 1) — Clientes premium
2. **ADULTO_MAYOR** (prioridad = 2) — Adultos mayores de 65 años
3. **REGULAR** (prioridad = 3) — Clientes estándar

Si dos clientes tienen la misma prioridad, atender al que llegó primero (FIFO).

Especificaciones técnicas:

- Usar un heap (árbol binario balanceado) para mantener la cola ordenada por prioridad
- Operaciones: agregar() y extraer() deben ejecutarse en O(log n)
- Implementar correctamente el proceso de "bubble up" (al agregar) y "bubble down" (al extraer)

2.2 Simulación de Atención — 5 pts

- **Procesar clientes** (3 pts)
    - Extraer clientes de la cola uno por uno respetando las prioridades
    - Simular el tiempo de atención: cada cliente tarda 2 segundos por producto (puede ser un print con delay o solo un cálculo)
    - Mostrar en tiempo real:
    - Atendiendo a: Juan Pérez (VIP, 5 productos)Tiempo de atención: 10 segundosCliente atendido correctamente
- **Agregar nuevos clientes durante la simulación** (2 pts)
    - El menú debe permitir agregar clientes mientras la simulación está en curso
    - Los nuevos clientes se insertan en la cola y serán atendidos según su prioridad
    - Mostrar: "Cliente María López agregado a la cola. Posición: 3"
- **Mostrar estado de la cola**
    - Listar todos los clientes en espera con su orden de atención

Ejemplo:

```
=== CLIENTES EN ESPERA ===
Posición | Nombre        | Tipo        | Productos
---------|---------------|-------------|----------
1        | Juan Pérez    | VIP         | 5
2        | María López   | ADULTO_MAYOR| 3
3        | Carlos Gómez  | REGULAR     | 8
```

3. Interfaz de Usuario — 5 pts

Crear un menú interactivo en consola con las siguientes opciones:

========== SISTEMA DE SUPERMERCADO ==========

--- GESTIÓN DE INVENTARIO ---

1. Insertar nuevo producto

2. Buscar producto por código

3. Actualizar producto

4. Eliminar producto

5. Mostrar inventario completo

--- GESTIÓN DE CLIENTES ---

6. Agregar cliente a la cola

7. Atender siguiente cliente

8. Ver clientes en espera

9. Simular atención de todos los clientes

--- OTROS ---

10. Mostrar estadísticas del sistema

11. Salir

Seleccione una opción:

Requisitos de la interfaz:

- Validación de entrada (manejo de excepciones para entradas inválidas)
- Mensajes claros y amigables para el usuario
- Mostrar confirmaciones de operaciones realizadas
- Permitir volver al menú principal desde cualquier opción
- Uso de colores o separadores visuales para mejorar legibilidad (opcional pero recomendado)

4. Estructura del Código y Documentación — 5 pts

4.1 Diseño de Clases — 2 pts

Organizar el código con las siguientes clases (mínimo requerido):

- Producto — representa un producto del supermercado
- NodoAVL — nodo del árbol AVL
- ArbolAVL — implementación del árbol con todas sus operaciones
- Cliente — representa un cliente en la cola
- ColaPrioridad — implementación de la cola de prioridad
- Supermercado — clase principal que integra ambas estructuras
- Menu — maneja la interfaz y entrada del usuario

Usar principios de POO:

- Encapsulación (atributos privados, getters/setters)
- Herencia (si es apropiado)
- Abstracción (métodos claros y bien definidos)
- No repetir código (DRY - Don't Repeat Yourself)

4.2 Documentación y Comentarios — 2 pts

- Comentarios al inicio de cada clase explicando su propósito
- Comentar métodos complejos (especialmente rotaciones del AVL y operaciones del heap)
- Usar Javadoc para documentar métodos públicos:
- /** * Inserta un producto en el árbol AVL manteniendo el balance * @param producto el producto a insertar * @return true si la inserción fue exitosa, false si el código ya existe */public boolean insertar(Producto producto) { ... }
- Comentarios claros sobre la lógica de rebalanceo y prioridades

4.3 Diagrama de Clases — 1 pt

Entregar un diagrama UML que muestre:

- Las clases principales y sus atributos
- Los métodos públicos de cada clase
- Puede ser en formato .png, .jpg, .pdf o un documento separado (recomendado usar herramientas como Draw.io, Lucidchart o PlantUML)

CRITERIOS DE EVALUACIÓN:

Criterio	Pts	Observaciones
Árbol AVL correcto y equilibrado	10	Rotaciones correctas, búsqueda eficiente
Operaciones de inventario	10	Todas las 5 operaciones funcionan correctamente
Cola de Prioridad correcta	5	Inserción y extracción respetan prioridades
Simulación de atención	5	Orden correcto, manejo de nuevos clientes
Interfaz de usuario	5	Menú intuitivo, validación de entrada
Diseño de clases	2	Organización clara, POO correctamente aplicado
Documentación	2	Comentarios y Javadoc adecuados
Diagrama UML	1	Claro y representativo
TOTAL	**40**	

REQUERIMIENTOS TÉCNICOS GENERALES:

- **Lenguaje:** Java (versión 8 o superior)
- **No se permite** usar librerías externas para Árbol AVL o Cola de Prioridad — deben implementarse desde cero
- Sí se puede usar ArrayList, LinkedList, Scanner, etc. de la librería estándar de Java y todo lo que quieran utilizar.
- El código debe compilar y ejecutarse sin errores

FORMA DE ENTREGA:

1. Archivo .zip o repositorio Git con todos los archivos .java
2. Archivo README.txt con instrucciones de compilación y ejecución
3. Diagrama UML de clases (archivo separado)
4. Evidencia de pruebas (screenshot del programa ejecutándose)
5. Comentarios en el código explicando decisiones de diseño