Taller Estructuras de Datos Dinámicos Lineales

Ejercicio 1: Listas doblemente enlazadas

Temática: Gestión de vehículos de vehículos en un taller mecánico

Enunciado
Un taller mecánico atiende los vehículos de acuerdo al orden de llegada y permite reorganizar la fila según prioridades (por ejemplo, emergencias). Cada vehículo tiene una placa, modelo y propietario.

El sistema debe permitir:

- Ingresar autos al inicio o al final de la fila.
- Insertar un vehículo en cualquier posición de la fila (por prioridad).
- Eliminar el primer, último o cualquier auto específico por placa.
- Recorrer la fila desde la recepción hasta la salida y viceversa, imprimiendo los datos.

Implemente la estructura de la lista doblemente enlazada con las operaciones mencionadas y cree un menú simple de consola para interactuar con la lista.

Guía de desarrollo

1. Análisis del problema:
El objetivo es organizar vehículos en el taller donde cada carro tiene una placa, modelo y propietario. La lista debe permitir reordenar vehículos y procesarlos en cualquier dirección.

2. Diseño de datos:

- Cree una clase Vehiculo para almacenar los datos de cada auto.
- Cree una clase Nodo que tenga atributos: Vehiculo, siguiente y anterior.
- Desarrolle una clase ListaDobleEnlazada para gestionar los nodos y proporcionar los métodos de inserción, eliminación y recorrido.

3. Operaciones requeridas:

- Agregar vehículo al inicio.
- Agregar vehículo al final.
- Insertar vehículo en posición (por prioridad).
- Eliminar vehículo al inicio.
- Eliminar vehículo al final.
- Eliminar vehículo por placa específica.
- Recorrer la lista en ambos sentidos (de inicio a fin y viceversa).

4. Interfaz de usuario:

- Use un menú en consola para ejecutar las operaciones anteriores.
- Solicite datos de vehículos al usuario por teclado.

5. Validación:

- Verificar condiciones como lista vacía, posición fuera de rango, placa no encontrada.
- Imprimir mensajes claros tras cada operación.
- Impresión de lista de vehículos para atención cuando se requiera

Nota: no usar clases de colección prediseñadas de Java

Ejercicio 2: Pilas

Temática: Gestión de envíos en una empresa de paquetería

Enunciado:
Una empresa de mensajería organiza diariamente los paquetes que deben ser cargados en un camión. Debido al espacio limitado, los paquetes se apilan unos sobre otros dentro del vehículo, de manera que el último en ingresar será el primero en ser descargado (principio LIFO).

Cada paquete tiene los siguientes datos:

- Código del paquete (entero)
- Destino (String)
- Peso en kilogramos (double)

El sistema deberá permitir:

1. Registrar un nuevo paquete en la pila (operación push).
2. Retirar el último paquete agregado de la pila (operación pop).
3. Consultar el paquete en la cima sin retirarlo (operación peek).
4. Mostrar todos los paquetes apilados actualmente.
5. Verificar si la pila está vacía.
6. Vaciar toda la pila (eliminar todos los elementos).
7. Salir del programa.

Guía de desarrollo

1. Análisis del problema
El objetivo es simular el apilamiento de paquetes dentro de un camión. Cada vez que se ingresa un paquete nuevo, debe ubicarse encima de los anteriores. Al retirar, se extrae el último ingresado. Esta lógica debe implementarse utilizando una pila construida manualmente, sin clases de colección prediseñadas en NetBeans.

2. Diseño de datos

- Cree una clase Paquete con atributos: codigo, destino y peso.
- Cree una clase Nodo que contenga un objeto Paquete y una referencia al siguiente nodo.
- Cree una clase PilaPaquetes para gestionar los nodos y las operaciones básicas (*push*, *pop*, *peek*, *vacio*, *eliminar*, *mostrar*).

3. Operaciones requeridas

- Apilar paquete: método push(Paquete p) agrega un nuevo nodo en la cima.
- Desapilar paquete: método pop() elimina y devuelve el paquete del tope.
- Ver cima: método peek() retorna el paquete del tope sin removerlo.
- Verificar vacía: método vacio() retorna verdadero si no hay elementos**.**
- Vaciar pila: método eliminar() elimina todos los paquetes**.**
- Mostrar pila: método mostrar() recorre e imprime los paquetes desde la cima hasta la base**.**

4. Interfaz de usuario

- Implemente una clase Main que presente un menú en consola con las operaciones anteriores.
- Solicite los datos del paquete al usuario mediante Scanner.

5. Validación

- Mensaje si se intenta desapilar o consultar una pila vacía.
- Mensaje de confirmación tras cada operación exitosa.
- Mostrar lista legible de todos los paquetes al imprimir la pila.

Nota: No se permite el uso de clases prediseñadas de Java como Stack, LinkedList o ArrayList. El ejercicio debe implementarse completamente usando nodos y/o referencias en memoria; mediante arreglos o listas enlazadas.

Ejercicio 3: Colas

Temática: Atención de clientes en una panadería

Enunciado:

Una panadería local requiere implementar la atención de sus clientes durante el día para optimizar el servicio. Los clientes son atendidos en el mismo orden en que llegan, por lo que el sistema debe seguir el principio FIFO (First In, First Out): el primer cliente en ingresar es el primero en ser atendido.

Cada cliente tiene los siguientes datos:

- Número de turno (int)
- Nombre del cliente (String)
- Pedido realizado (String)

El sistema debe permitir:

- Registrar la llegada de un cliente (ingresar al final de la cola).
- Atender al cliente del frente (retirar el primero en la cola).
- Consultar quién es el siguiente cliente en ser atendido (mostrar el frente sin eliminarlo).
- Mostrar todos los clientes que están esperando en orden de atención.
- Verificar si la cola está vacía.
- Vaciar completamente la cola.
- Salir del programa.

Guía de desarrollo

1. Análisis del problema

El objetivo es controlar la cola de atención de una panadería, donde cada cliente espera su turno según su orden de llegada. La cola debe ser implementada utilizando referencias enlazadas, no estructuras de colección prediseñadas.

2. Diseño de datos

- Crear una clase Cliente con atributos: numeroTurno, nombre, y pedido.
- Crear una clase Nodo que contenga un objeto Cliente y una referencia al siguiente nodo.
- Crear una clase ColaClientes que controle las operaciones sobre la cola (agregar, eliminar, ver frente, verificar vacía, mostrar todos).

3. Operaciones requeridas

- Encolar cliente: método encolar(Cliente c) agrega un nuevo nodo al final.
- Desencolar cliente: método desencolar() elimina y devuelve el nodo al frente.
- Ver frente: método verFrente() muestra el siguiente cliente sin retirarlo.
- Verificar vacía: método estaVacia() devuelve verdadero si no hay clientes.
- Vaciar cola: método vaciar() elimina todos los clientes.
- Mostrar cola: método mostrar() recorre la cola desde el frente hasta el final, imprimiendo los datos de cada cliente.

4. Interfaz de usuario

- Implementar una clase Main con un menú de consola interactivo que permita ejecutar las operaciones anteriores.
- Solicitar datos al usuario mediante Scanner.
- Imprimir mensajes informativos tras cada acción.

5. Validación

- Mostrar un mensaje cuando se intente atender o consultar en una cola vacía.
- Confirmar al usuario cada ingreso y salida de clientes.
- Imprimir la lista completa de clientes pendientes cuando se solicite.

Nota: No utilice las clases Queue, LinkedList ni ninguna colección de Java. Todas las operaciones deben implementarse manualmente mediante nodos y referencias en memoria, mediante arreglos o listas enlazadas.

Evidencias de entrega

- Documento pdf con la solución y la impresión de los pantallazos del manejo de la interfaz de usuario, menú, resultado de opciones
- Archivo comprimido con el proyecto o proyectos
- El archivo comprimido debe etiquetarse de la siguiente manera: Apellido_Nombre_Código