Simulación de Lista de Tareas con Lista Enlazada en Java

1. Introducción

Este documento presenta la solución al ejercicio de implementación de una lista de tareas (To-Do List) utilizando una lista enlazada simple implementada manualmente en Java.

El objetivo es fortalecer los conocimientos de estructuras de datos básicas como listas enlazadas, la programación orientada a objetos, y el uso de enumeraciones.

Se resuelven y explican las operaciones requeridas: agregar, eliminar, listar, modificar estado y ordenar.

---

2. Temas que se trabajarán

Estructuras de datos: Lista enlazada simple.

Programación orientada a objetos: Encapsulamiento, clases y métodos.

Enumeraciones en Java: Para prioridad y estado de tareas.

Lógica de ordenamiento: Ordenar nodos por prioridad.

Diseño modular: Separación clara de responsabilidades.

3. Diseño y estructura de clases

Enum Estado: Define si una tarea está PENDIENTE o COMPLETADA.

Enum Prioridad: ALTA, MEDIA, BAJA.

Clase Tarea: Contiene los datos de cada tarea.

Clase Nodo: Representa un nodo de la lista enlazada.

Clase ListaTareas: Administra los nodos y operaciones.

Clase Usuario: Método main para pruebas.

La lógica se organiza para que cada clase tenga una responsabilidad clara y única.

---

4. Solución en código (con explicación)

```
enum Estado { PENDIENTE, COMPLETADA }
enum Prioridad { ALTA, MEDIA, BAJA }

class Tarea {
```

```
    String id, descripcion;
    Estado estado;
    Prioridad prioridad;

    public Tarea(String id, String descripcion, Prioridad prioridad) {
        this.id = id;
        this.descripcion = descripcion;
        this.estado = Estado.PENDIENTE;
        this.prioridad = prioridad;
    }

    public String toString() {
        return id + ": " + descripcion + " [" + prioridad + ", " + estado + "]";
    }
}

class Nodo {
    Tarea tarea;
    Nodo siguiente;

    public Nodo(Tarea tarea) {
        this.tarea = tarea;
        this.siguiente = null;
    }
}
```

```
class ListaTareas {
  Nodo cabeza;

  public void agregar(String id, String desc, Prioridad p) {
    Nodo nuevo = new Nodo(new Tarea(id, desc, p));
    if (cabeza == null) cabeza = nuevo;
    else {
      Nodo temp = cabeza;
      while (temp.siguiente != null)
        temp = temp.siguiente;
      temp.siguiente = nuevo;
    }
  }

  public boolean eliminar(String id) {
    if (cabeza == null) return false;
    if (cabeza.tarea.id.equals(id)) {
      cabeza = cabeza.siguiente;
      return true;
    }
    Nodo temp = cabeza;
    while (temp.siguiente != null && !temp.siguiente.tarea.id.equals(id))
      temp = temp.siguiente;
    if (temp.siguiente == null) return false;
    temp.siguiente = temp.siguiente.siguiente;
    return true;
```

```
}

public boolean marcarCompletada(String id) {
   Nodo temp = cabeza;
   while (temp != null) {
      if (temp.tarea.id.equals(id)) {
         temp.tarea.estado = Estado.COMPLETADA;
         return true;
      }
      temp = temp.siguiente;
   }
   return false;
}

public void listar(boolean soloPendientes) {
   Nodo temp = cabeza;
   while (temp != null) {
      if (!soloPendientes || temp.tarea.estado == Estado.PENDIENTE)
         System.out.println(temp.tarea);
      temp = temp.siguiente;
   }
}

public void ordenarPorPrioridad() {
   if (cabeza == null || cabeza.siguiente == null) return;
   Nodo actual = cabeza, index = null;
```

```
        while (actual != null) {
            index = actual.siguiente;
            while (index != null) {
                if (index.tarea.prioridad.ordinal() < actual.tarea.prioridad.ordinal()) {
                    Tarea temp = actual.tarea;
                    actual.tarea = index.tarea;
                    index.tarea = temp;
                }
                index = index.siguiente;
            }
            actual = actual.siguiente;
        }
    }
}

public class Usuario {
    public static void main(String[] args) {
        ListaTareas lista = new ListaTareas();
        lista.agregar("1", "Estudiar Java", Prioridad.ALTA);
        lista.agregar("2", "Leer libro", Prioridad.MEDIA);
        lista.agregar("3", "Ejercicio físico", Prioridad.BAJA);

        lista.marcarCompletada("2");
        lista.ordenarPorPrioridad();

        System.out.println("--- Tareas Pendientes ---");
```

```
        lista.listar(true);
    }
}
```

5. Conclusión y beneficios

Este ejercicio enseña a implementar estructuras de datos desde cero, entender cómo se enlazan nodos, cómo gestionar inserciones y eliminaciones, y cómo aplicar ordenamientos personalizados.

El uso de enumeraciones enriquece la semántica del código, y todo se logra sin depender de clases preconstruidas como ArrayList.

¿Por qué?

Porque aprender cómo funcionan internamente las estructuras de datos te permite resolver problemas complejos con eficiencia y control total.

¿Cuándo?

Cuando se requiere una estructura ligera, personalizada, o se está aprendiendo algoritmos fundamentales.

¿Dónde?

En sistemas donde el control manual sobre memoria y estructura sea necesario, como aplicaciones embebidas o educativas.

---

6. Bibliografía (Formato APA)

Oracle. (2024). The Java Tutorials. Recuperado de https://docs.oracle.com/javase/tutorial/

Horstmann, C. S. (2019). Core Java Volume I--Fundamentals (11th ed.). Prentice Hall.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Java (6th ed.). Wiley.