# PROYECTO

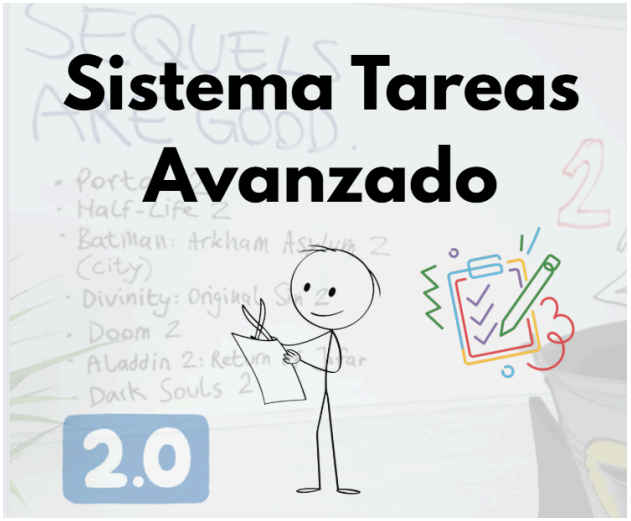
## Estructuras de Datos

Nombre: Isaac Alejandro Isaias Betance
Materia: Estructura de Datos
Actividad: Proyecto Final

# SistemaTareasAvanzado

## Introduccion

Este proyecto lo hice pensando en un sistema de tareas que se puede usar en la vida real para organizar cosas de la escuela o del trabajo, se mete la tarea , se guarda con prioridad , con fecha y hasta dependencias entre ellas. Tambien se pueden registrar empleados en un arbol binario para buscarlos mas rapido. La idea es que sea un sistema sencillo donde tengas recordatorios de tareas que guardes y ordenarlas por prioridad.




## 1. Pilas , colas y listas

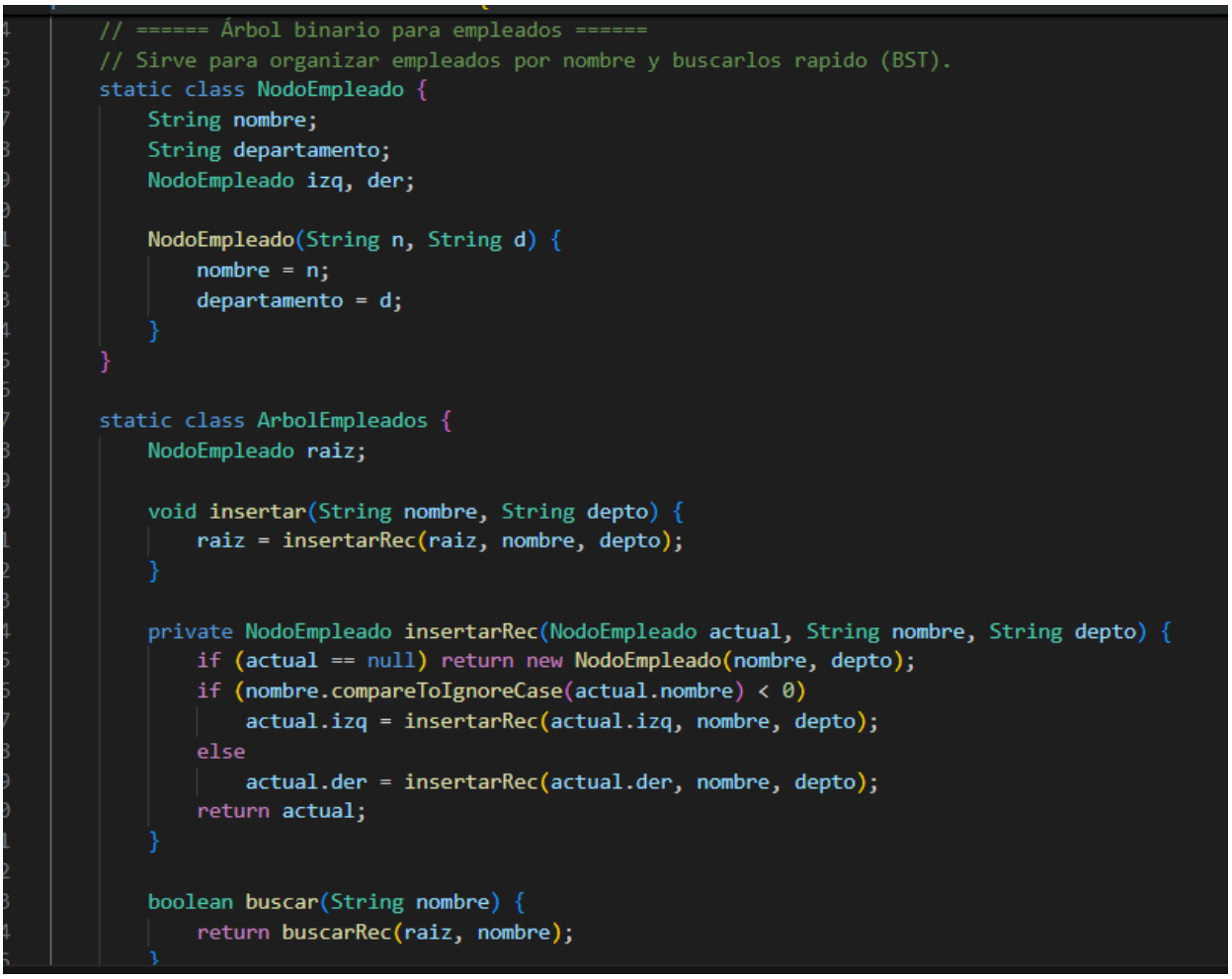
Implemente una **cola de prioridad** que organiza las tareas segun su urgencia y fecha . Asi primero salen las mas importantes. Tambien use listas para ordenar y mostrar los datos.(en los comentarios de el codigo explico como funciona).

```
// ====== Cola de prioridad (urgencia + fecha) ======
// Usada para seleccionar primero lo mas urgente y si hay empate, lo con fecha mas cercana.
static class ColaPrioridad {
    PriorityQueue<Tarea> pq = new PriorityQueue<>((a, b) -> {
        int cmp = Integer.compare(b.urgencia, a.urgencia);
        if (cmp == 0) return a.fechaEntrega.compareTo(b.fechaEntrega);
        return cmp;
    });

    void add(Tarea t) { pq.offer(t); }
    Tarea poll() { return pq.poll(); }
    boolean isEmpty() { return pq.isEmpty(); }
    List<Tarea> toList() { return new ArrayList<>(pq); }
}
```

## 2. Arbol binario

Use un arbol binario de busqueda pa guardar empleados con su departamento . Esto deja buscarlos rapido por nombre y mostrarlos en orden alfabetico.



```java
// ====== Árbol binario para empleados ======
// Sirve para organizar empleados por nombre y buscarlos rapido (BST).
static class NodoEmpleado {
    String nombre;
    String departamento;
    NodoEmpleado izq, der;

    NodoEmpleado(String n, String d) {
        nombre = n;
        departamento = d;
    }
}

static class ArbolEmpleados {
    NodoEmpleado raiz;

    void insertar(String nombre, String depto) {
        raiz = insertarRec(raiz, nombre, depto);
    }

    private NodoEmpleado insertarRec(NodoEmpleado actual, String nombre, String depto) {
        if (actual == null) return new NodoEmpleado(nombre, depto);
        if (nombre.compareToIgnoreCase(actual.nombre) < 0)
            actual.izq = insertarRec(actual.izq, nombre, depto);
        else
            actual.der = insertarRec(actual.der, nombre, depto);
        return actual;
    }

    boolean buscar(String nombre) {
        return buscarRec(raiz, nombre);
    }
```

## 3. Algoritmo recursivo

El arbol binario se recorre de forma recursiva con inOrder . Eso ayuda a mostrar los empleados sin tener que hacer ciclos complicados.

```java
// ====== Funciones de gestion de tareas
static void addTask() {
    System.out.print(s:"Titulo: ");
    String tit = sc.nextLine();
    System.out.print(s:"Depto: ");
    String depto = sc.nextLine();
    int urg = readInt(prompt:"Urgencia (1-5): ", min:1, max:5);
    // generamos fecha aproximada sumando dias segun urgencia (demo)
    Date fecha = new Date(System.currentTimeMillis() + urg * 86400000L);

    Tarea t = new Tarea(tit, depto, urg, fecha);
    // lo metemos en cola prioridad y en la base de datos hash
    cola.add(t);
    db.addTarea(t);

    System.out.println("Tarea agregada: " + t);

    // opcional: preguntar dependencias al crear
    System.out.print(s:"¿Tiene dependencias? (s/n): ");
    String ans = sc.nextLine().trim();
    if (ans.equalsIgnoreCase(anotherString:"s")) {
        System.out.print(s:"Introduce ID(s) separadas por comas (ej: 1,2): ");
        String line = sc.nextLine().trim();
        String[] parts = line.split(regex:",");
        for (String p : parts) {
            try {
                int depId = Integer.parseInt(p.trim());
                if (db.getTarea(depId) != null) grafo.addDep(t.id, depId);
            } catch (Exception e) { /* skip invalid */ }
```

# 4. Algoritmo de divide y venceras

Para buscar tareas por ID hice una **busqueda binaria**. Primero ordena la lista y luego divide el rango hasta encontrar la tarea .

```java
    // búsqueda binaria por id (divide y vencerás)
    Tarea buscarPorID(int id) {
        lista.sort(Comparator.comparingInt(t -> t.id));
        int izq = 0, der = lista.size() - 1;
        while (izq <= der) {
            int mid = (izq + der) / 2;
            if (lista.get(mid).id == id) return lista.get(mid);
            if (lista.get(mid).id < id) izq = mid + 1;
            else der = mid - 1;
        }
        return null;
    }

    void mostrar() {
        if (lista.isEmpty()) {
            System.out.println(x:"lista vacia.");
            return;
        }
        lista.forEach(System.out::println);
    }
}
```

# 5. Algoritmo de ordenamiento

Implemente ordenamiento de tareas por **urgencia** y por **fecha** usando comparadores . Asi se puede decidir como ver la lista segun lo que mas convenga.

```java
    void ordenarPorUrgencia() {
        // Orden descendente por urgencia (5 -> 1)
        lista.sort((a, b) -> Integer.compare(b.urgencia, a.urgencia));
    }

    void ordenarPorFecha() {
        // Orden ascendente por fecha (antes primero)
        lista.sort(Comparator.comparing(t -> t.fechaEntrega));
    }
```

```
--- Cola de prioridad ---
[1] Deploy servidor (Dept: TI, urg: 5, fecha: Thu Sep 25 12:46:39 CST 2025, estado: pendiente)
[2] Pagar renta (Dept: Hogar, urg: 5, fecha: Tue Sep 30 12:22:48 CST 2025, estado: pendiente)
[3] Auditoria interna (Dept: Finanzas, urg: 4, fecha: Sun Sep 28 12:46:39 CST 2025, estado: pendiente)
[2] Disenar logo (Dept: Marketing, urg: 2, fecha: Fri Oct 03 12:46:39 CST 2025, estado: pendiente)
[1] Pagar impuestos 4 (Dept: Hogar, urg: 4, fecha: Mon Sep 29 12:22:08 CST 2025, estado: pendiente)
[3] Comprar leche (Dept: Hogar, urg: 3, fecha: Sun Sep 28 12:23:17 CST 2025, estado: pendiente)
```

# 6. HashMap

El HashMap lo use para guardar todas las tareas y empleados . Sirve para acceso rapido cuando se busca por ID o nombre.

```java
// marcar tarea completada por ID (busca en la BaseDatos)
static void marcarTareaCompletada() {
    int id = readInt(prompt:"ID de la tarea a marcar completada: ", min:1, Integer.MAX_VALUE);
    Tarea t = db.getTarea(id);
    if (t == null) System.out.println(x:"No existe tarea con ese ID.");
    else {
        t.marcarCompletada();
        System.out.println("Tarea marcada como completada: " + t);
    }
}

// eliminar tarea: la quitamos del hashmap y la cola (si aparece), y del grafo (limpiamos deps)
static void eliminarTarea() {
    int id = readInt(prompt:"ID de la tarea a eliminar: ", min:1, Integer.MAX_VALUE);
    Tarea t = db.getTarea(id);
    if (t == null) { System.out.println(x:"No existe tarea con ese ID."); return; }
    // quitar del hashmap
    db.tareas.remove(id);
    // quitar de la cola: reconstruimos pq sin esa tarea
    List<Tarea> tmp = cola.toList();
    tmp.removeIf(x -> x.id == id);
    cola = new ColaPrioridad();
    for (Tarea x : tmp) cola.add(x);
    // quitar dependencias que apuntan o salen de la tarea
    // removemos la entrada y tambien borramos referencias en listas
    grafo.adj.remove(id);
    for (List<Integer> deps : grafo.adj.values()) deps.removeIf(d -> d == id);
    System.out.println("Tarea eliminada de todas las estructuras: " + t.titulo);
}
```

# 7. Grafo

Implemente un grafo de dependencias de tareas . Eso deja decir que una tarea no se puede hacer hasta que se termine otra . Ejemplo: el deploy depende de la auditoria.

```java
// ====== PERSISTENCIA: guardar y cargar estado (db + grafo) ======

// guarda estado a disco (db y grafo)
static void guardarEstado() {
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(ARCHIVO_DB))) {
        oos.writeObject(db);
        System.out.println("Base de datos guardada: " + ARCHIVO_DB);
    } catch (Exception e) {
        System.out.println("Error guardando DB: " + e.getMessage());
    }
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(ARCHIVO_GRAFO))) {
        oos.writeObject(grafo);
        System.out.println("Grafo guardado: " + ARCHIVO_GRAFO);
    } catch (Exception e) {
        System.out.println("Error guardando grafo: " + e.getMessage());
    }
}

// carga estado de disco (si existe)
static void cargarEstado() {
    File fdb = new File(ARCHIVO_DB);
    if (fdb.exists()) {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fdb))) {
            BaseDatos loaded = (BaseDatos) ois.readObject();
            if (loaded != null) {
                db = loaded;
                // reconstruir cola desde db (para mantener consistencia)
                cola = new ColaPrioridad();
                for (Tarea t : db.tareas.values()) cola.add(t);
                System.out.println("Base de datos cargada con " + db.tareas.size() + " tareas.");
            }
```

## Conclusion

Este sistema cumple con lo que pedia el proyecto y hasta mas , porque guarde datos en archivos y permite eliminarlos o marcarlos como completados. Para mejorarlo todavia mas podriamos hacer una app movil y subirla a la play store que mande notificaciones de recordatorios tipo pagar renta , pagar impuestos o esas cosas de la vida.

Tambien cambie el nombre de el proyecto de el avance al proyecto final: