



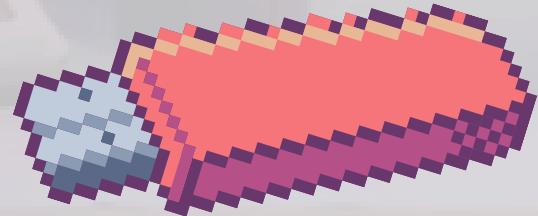
Proyecto Final – Sistema Tareas Avanzado

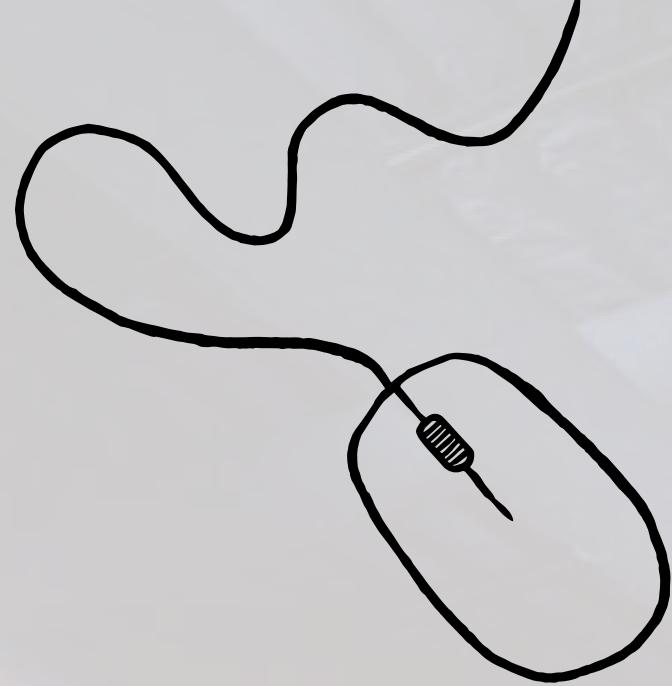
Proyecto Final: Sistema de Gestión de Tareas

Nombre: Isaac Alejandro Isaias Betance

Materia: Estructura de Datos

Este proyecto lo arme como parte de la materia , buscando que no solo fuera código sino tambien algo que simulara un sistema real que si podria usarse.





Introducción

Yo hice un java para organizar tareas , usando varias estructuras de datos.
Lo pense como si fuera un sistema que te ayuda tanto en la oficina como en cosas de la vida personal.
La idea fue crear algo que se sintiera util y sin ser muy complejo .



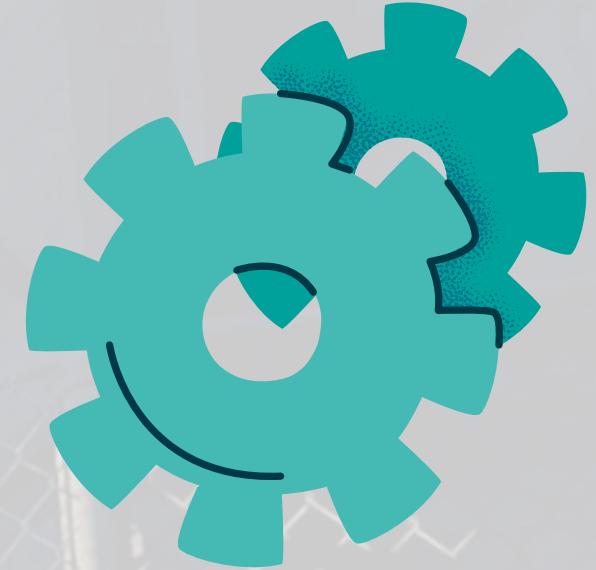
Sistema Tareas Avanzado

- Portal 2
- Half-Life 2
- Batman: Arkham Asylum 2 (City)
- Divinity: Original Sin 2
- Doom 2
- Aladdin 2: Return to Agrabah
- Dark Souls 2

2.0

An illustration of a simple stick figure holding a clipboard with checkmarks, standing next to a notepad and a pencil, set against a background of a whiteboard with handwritten notes like 'ARE GOOD.' and '2'.

Avance del Proyecto

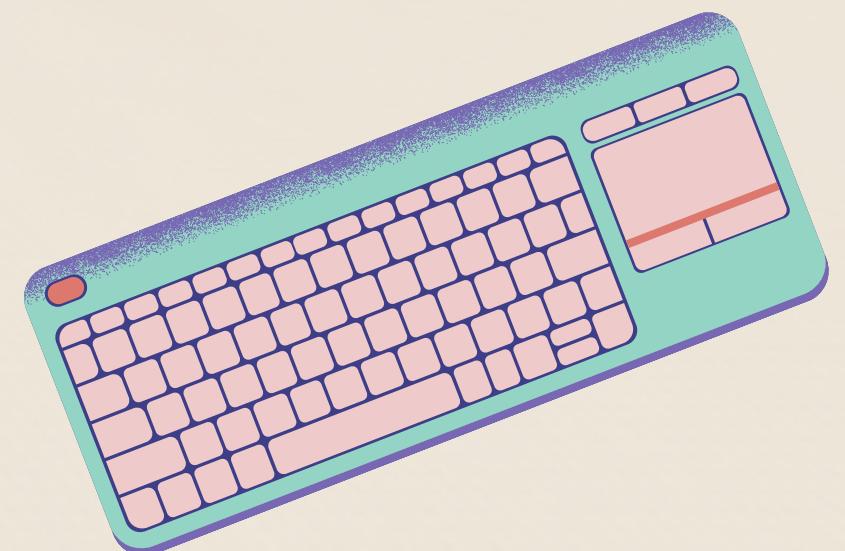


En el avance solo tenia pilas , colas y listas
ahi probe como meter , eliminar y ver tareas , fue como el cimiento



Proyecto Final

Integre lo nuevo :
Cola de prioridad
Arbol binario
HashMap
Grafo de dependencias
Con eso el sistema se volvio mas completo.





Fundamentos Usados

Me base en estos fundamentos:
Pilas (LIFO) para lo urgente
Colas (FIFO) para lo ordenado
Arboles binarios para busqueda rapida
Recursividad y divide y vencerás pa problemas grandes
Hash para acceso veloz
Grafos para dependencias
Cada uno fue pensado en donde encajaba mejor.



Implemente una **cola de prioridad**

- 1-. que organiza las tareas segun su urgencia y fecha .

```
// ===== 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); }
}
```

```
// ===== 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);
    }
}
```

- 2-. **Arbol binario** de busqueda de busqueda para guardar empleados con su departamento . Esto deja buscarlos rapido por nombre y mostrarlos en orden alfabetico.



```
// ===== Funciones de gestion de tareas
static void addTask() {
    System.out.print(s:"Titulo: ");
    String tit = sc.nextLine();
    System.out.print(s:"Dept: ");
    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);
```

- 4-. Buscar tareas por ID hice una busqueda binaria. Primero ordena la lista y luego divide el rango hasta encontrar la tarea .

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

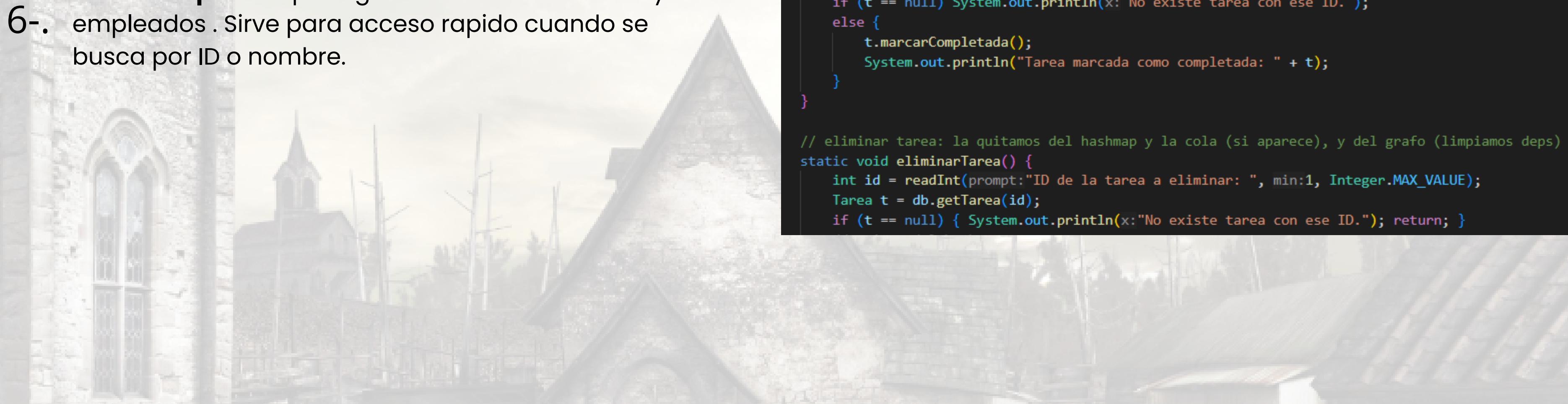
void ordenarPorFecha() {
    // Orden ascendente por fecha (antes primero)
    lista.sort(Comparator.comparing(t -> t.fechaEntrega));
```

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

```
// 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;
```

- 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.

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



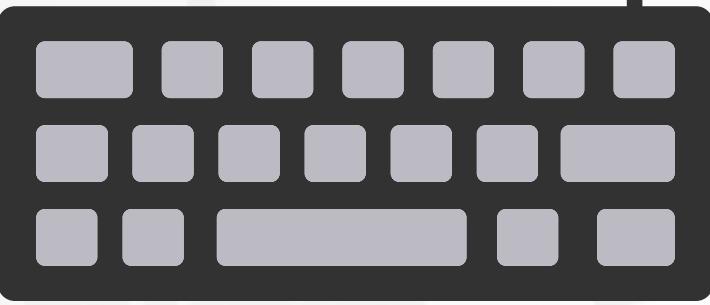
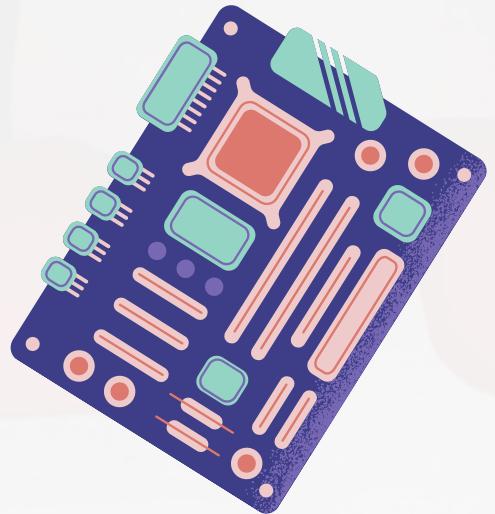
```
// marcar tarea completada por ID (busca en la BaseDatos)
static void marcarTareaCompletada() {
    int id = readInt(prompt:"ID de la tarea a marcar completada: ", min:1, Integer.MAX_VALUE);
    Tarea t = db.getTarea(id);
    if (t == null) System.out.println(x:"No existe tarea con ese ID.");
  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

```
// ===== 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(graf);
        System.out.println("Grafo guardado: " + ARCHIVO_GRAFO);
    } catch (Exception e) {
        System.out.println("Error guardando grafo: " + e.getMessage());
    }
}
```

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.



Estrategias Implementadas

Lo que hice fue:

Priorizar tareas segun urgencia y fecha , para que no se acumulen

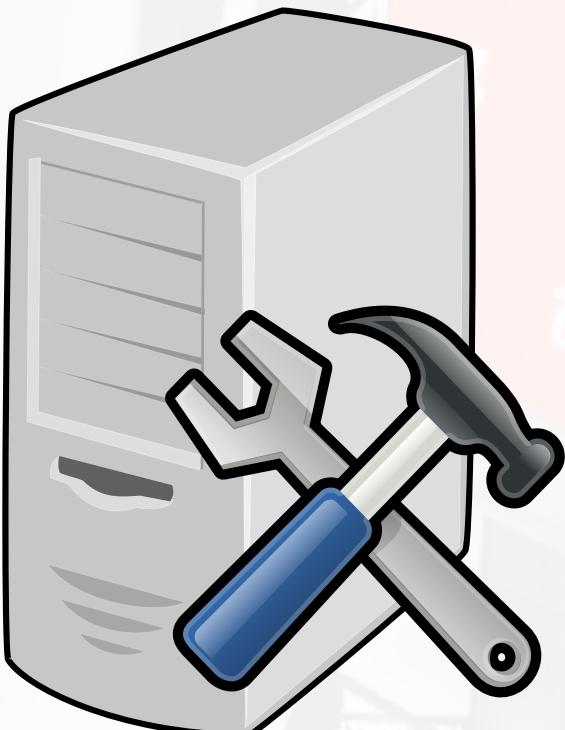
Buscar empleados rapido en arbol segun su depto

Ordenar tareas con algoritmos para que se vea claro que va primero

Guardar datos en HashMap para accederlos mas facil

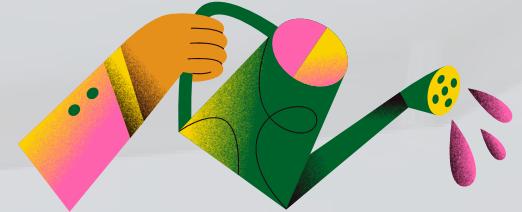
Mostrar dependencias con grafo para proyectos grandes

Asi cada parte del sistema cumple con algo concreto.





Ejemplos Practicos



Cola: pagar recibo antes que corten la luz , porque eso no puede esperar

Arbol: buscar rapido un empleado en depto sin andar revisando todo

Grafo: cuando una tarea depende de otra como pruebas antes del deploy

HashMap: buscar tarea x ID al instante sin perder tiempo

Lo hice pensando en ejemplos que podrian usarse en el programa.



Impacto del Proyecto



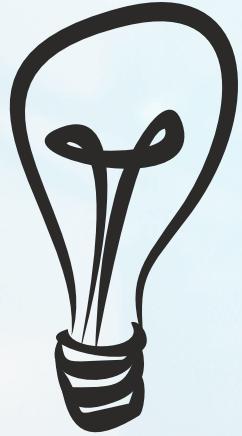
Este sistema puede ayudar a que la chamba se organice mejor y que no se olviden cosas importantes.

Tambien podria servir para proyectos grandes donde hay muchas dependencias.
Incluso para la vida personal con recordatorios sencillos.





Reflexion



Aprendi a combinar varias estructuras en un mismo sistema y vi como trabajan juntas. Entendi que no solo es teoria de libro , sino que se puede aplicar directo en la practica. Tambien me hizo mejorar en como organizo el codigo y en mi logica.



Conclusión



- Permite guardar tareas en archivos, eliminarlas o marcarlas como completadas.
- Se integraron estructuras avanzadas que lo hacen rápido y útil en la vida real.
- Como mejora futura, se podría crear una app móvil y subirla a la Play Store.
 - Así mandaría notificaciones de recordatorios tipo pagar renta, impuestos o pendientes personales.
- El proyecto demuestra que la teoría de estructuras de datos se puede aplicar directo a la práctica.

