gestion de tareas

nombres: Ezequiel Mermet, Leonardo Zavala y Valentín Lucero

📘 Descripcion general

Este proyecto consiste en una aplicación de consola desarrollada en Java que permite al usuario **gestionar tareas**, almacenándolas en una **tabla hash** implementada manualmente. Las operaciones disponibles son:

- Crear una tarea nueva.
- Buscar una tarea por nombre.
- Eliminar una tarea (baja lógica).
- Mostrar todas las tareas activas.

Cada tarea contiene un nombre único, una descripción, un estado (pendiente, en progreso o finalizada) y un rango de fechas (inicio y fin).

Arquitectura del sistema

El sistema está dividido en tres clases principales:

🔷 Main.java

Es la clase principal que gestiona la interacción con el usuario. Contiene un ciclo `do-while` con un menú que permite:

- **Crear una tarea:** Llama a `Tarea.cargarDesdeTeclado()` para recolectar datos del usuario y luego inserta la tarea en la tabla.
- **Buscar tarea por nombre:** Solicita un nombre y muestra los datos si la tarea está activa.
- **Eliminar tarea:** Da de baja lógicamente una tarea (no la borra físicamente).
- **Mostrar tareas:** Recorre y muestra las tareas activas en la tabla.
- **Salir del programa:** Finaliza el ciclo y cierra el escáner.

Este flujo centraliza el control del programa, mientras que la lógica de datos está encapsulada en las otras clases.

🔷 Tarea.java

Esta clase representa una **tarea individual**. Encapsula todos los atributos y comportamientos relacionados con una tarea:

- `codigo`: UUID único generado automáticamente.
- `nombre`: obligatorio y único (no se permiten duplicados).
- `descripcion`: opcional, con un límite de 200 caracteres.
- `estado`: valores posibles → 1 = pendiente, 2 = en progreso, 3 = finalizada.
- `fechaInicio` y `fechaFin`: fechas ingresadas por el usuario con validación.

- `esAlta`: indica si la tarea está activa (`true`) o fue dada de baja (`false`).}

Operaciones disponibles:

`Tarea()`

Qué hace:

Es el constructor por defecto. Se ejecuta automáticamente cuando se crea una nueva instancia de `Tarea`.

Lógica interna:

- Genera un `UUID` (identificador único) automáticamente para que cada tarea sea única.
- Marca la tarea como activa (`esAlta = true`).

Parámetros:

Ninguno.

Devuelve:

Nada (es un constructor).

`Tarea(String nombre, String descripcion, int estado)`

Qué hace:

Crea una tarea con datos iniciales específicos.

Lógica interna:

- Llama al constructor por defecto para generar el código.
- Asigna los valores recibidos a los atributos correspondientes.

Parámetros:

- `nombre`: nombre de la tarea.
- `descripcion`: texto descriptivo.
- `estado`: número entero que representa el estado (1 o 2).

Devuelve:

Nada (es un constructor).

`static Tarea cargarDesdeTeclado(TablaDispersa tabla)`

Qué hace:

Guía al usuario para crear una tarea ingresando los datos por consola.

Lógica interna:

- Pide nombre (obligatorio y único).
- Pide descripción (máx. 200 caracteres).
- Pide estado (solo 1 o 2).
- Pide fechas, y valida que:
  - La fecha de inicio no sea antes de hoy.
  - La fecha de fin no sea antes de la fecha de inicio.

- Valida formato y tipo de datos, usando `try/catch` para errores.
- Devuelve una nueva instancia de `Tarea`.

Parámetros:

- `tabla`: se usa para verificar si el nombre ingresado ya existe.

Devuelve:

Un objeto `Tarea` con los datos ingresados.

`String toString()`

Qué hace:

Muestra todos los datos de la tarea de manera legible para el usuario.

Lógica interna:

- Convierte el número del estado a texto ("pendiente", "en progreso", "finalizada").
- Devuelve un bloque de texto bien formateado con todos los campos.

Parámetros:

Ninguno.

Devuelve:

Un `String` con los datos de la tarea.

`boolean isAlta()`

Qué hace:

Indica si la tarea está activa (no eliminada).

Lógica interna:

Devuelve el valor del atributo `esAlta`.

Parámetros:

Ninguno.

Devuelve:

`true` si está activa, `false` si fue dada de baja.

`Getters y Setters`

Qué hacen:

Permiten obtener (`get`) y modificar (`set`) valores de los atributos privados.

Ejemplos:

- `getNombre()` → devuelve el nombre de la tarea.
- `setFechaFin(LocalDate fecha)` → cambia la fecha de fin de la tarea

---

3. Clase `TablaDispersa`

Implementa una tabla hash con las siguientes características:

- **Tamaño fijo**: 101 posiciones.
- **Resolución de colisiones**: mediante **sondeo cuadrático**, que evita ciclos y aglomeraciones.
- **Método de hash**: utiliza el **método de la multiplicación**, una técnica eficiente que toma una parte decimal del producto entre un valor numérico y una constante irracional (`A = 0.6180339887`, basada en el número áureo).

Operaciones disponibles:

`TablaDispersa()`

Qué hace:

Inicializa la tabla hash vacía.

Lógica interna:

- Crea un arreglo de `Tarea` de tamaño fijo (`101`).
- Inicializa contador de elementos y factor de carga.

Parámetros:

Ninguno.

Devuelve:

Nada (es un constructor).

`int calcularPosicion(String codigo)`

Qué hace:

Calcula la posición hash donde debería ir la tarea.

Lógica interna:

1. Llama a `obtenerValorNumerico(codigo)`

   → convierte texto en número (suma ASCII).
2. Multiplica por `A = 0.6180339887`

   → número irracional basado en el número áureo.
3. Toma solo la parte decimal del resultado

   → `% 1`
4. Multiplica por la capacidad (101)

   → Y redondea hacia abajo (floor).

Esto genera una posición de 0 a 100.

Parámetros:

- `codigo`: cadena UUID de la tarea.

Devuelve:

Un número entero entre 0 y 100 (posición en la tabla).

`private int obtenerValorNumerico(String codigo)`

Qué hace:

Convierte una cadena (`codigo`) en un valor numérico.

Lógica interna:

- Toma los primeros 10 caracteres del código.
- Suma los valores ASCII de cada carácter.
- Retorna esa suma como base para calcular el hash.

Parámetros:

- `codigo` : cadena UUID.

Devuelve:

Un número entero.

`int resolverColision(int posicionInicial, int i)`

Qué hace:

Calcula una nueva posición si hay una colisión, usando **sondeo cuadrático**.

Lógica interna:

- Usa la fórmula: `(posicionInicial + i²) % CAPACIDAD` .

Parámetros:

- `posicionInicial` : posición hash original.
- `i` : número de intento (1, 2, 3...).

Devuelve:

La nueva posición.

`boolean insertar(Tarea t)`

Qué hace:

Intenta insertar una tarea en la tabla.

Lógica interna:

- Calcula la posición usando `calcularPosicion()` .
- Si la posición está ocupada, aplica `resolverColision()` hasta encontrar espacio.
- Verifica que el código no esté repetido.
- Aumenta `numElementos` .
- Inserta la tarea y actualiza el factor de carga.

Parámetros:

- `t` : objeto `Tarea` a insertar.

Devuelve:

- `true` si la tarea se insertó correctamente.
- `false` si está duplicada o no hay espacio.

`Tarea buscarPorNombre(String nombre)`

Qué hace:

Busca una tarea activa por su nombre (ignora mayúsculas/minúsculas).

Lógica interna:

- Recorre el arreglo y compara los nombres con `equalsIgnoreCase()`.

Parámetros:

- `nombre`: nombre de la tarea.

Devuelve:

- Objeto `Tarea` si la encuentra.
- `null` si no está o está dada de baja.

`boolean eliminarPorNombre(String nombre)`

Qué hace:

Elimina una tarea por nombre, marcándola como inactiva (`esAlta = false`).

Lógica interna:

- Recorre la tabla.
- Si encuentra una tarea con ese nombre y está activa, la da de baja.

Parámetros:

- `nombre`: nombre de la tarea a eliminar.

Devuelve:

- `true` si la encontró y dio de baja.
- `false` si no la encontró.

`double calcularFactorCarga()`

Qué hace:

Calcula el porcentaje de la tabla que está ocupado.

Lógica interna:

- `numElementos / CAPACIDAD`

Parámetros:

Ninguno.

Devuelve:

Valor entre 0 y 1 (ej: 0.45 = 45% ocupado).

`void mostrarTabla()`

Qué hace:

Imprime todas las tareas activas de la tabla con su información y posición.

Lógica interna:

- Recorre el arreglo.
- Para cada posición con una tarea activa, muestra su nombre y llama a `toString()`.

Parámetros:

Ninguno.

Devuelve:

Nada (solo imprime en consola).

Validaciones y Manejo de Errores (con ejemplos)

Durante la carga y manipulación de datos, se implementaron validaciones estrictas para asegurar que la información ingresada por el usuario sea válida, coherente y segura. Además, se utilizó manejo de errores (`try/catch`) para evitar que el programa se detenga por entradas incorrectas.

✔️ Validaciones al crear una tarea (`cargarDesdeTeclado`)

- **Nombre**
  - **Regla:** No puede estar vacío ni repetido.
  - **Ejemplo:**

    Si el usuario presiona Enter sin escribir nada:

    → *"El nombre no puede estar vacío."*

    Si el usuario escribe un nombre que ya existe:

    → *"Ya existe una tarea con ese nombre. Ingrese un nombre diferente."*
- **Descripción**
  - **Regla:** Opcional, pero no debe superar los 200 caracteres.
  - **Ejemplo:**

    Si el usuario pega un texto muy largo:

    → *"La descripción es demasiado larga. Intente nuevamente."*
- **Estado**
  - **Regla:** Solo se aceptan los valores `1` (pendiente) o `2` (en progreso).
  - **Ejemplo:**

    Si el usuario ingresa `5` o escribe `pendiente` en vez de un número:

    → *"Estado inválido. Solo se permiten 1 o 2."*

    → *(capturado por `NumberFormatException`)*
- **Fecha de inicio**
  - **Regla:** Debe tener formato `YYYY-MM-DD` y no puede ser anterior a la fecha actual.
  - **Ejemplo:**

    Si el usuario escribe `12/05/2025`:

    → *"Fecha inválida. Intente de nuevo con formato YYYY-MM-DD."*

    Si escribe `2022-01-01`:

    → *"La fecha de inicio no puede ser anterior a hoy."*
- **Fecha de fin**

- **Regla:** Debe estar en formato válido y no puede ser anterior a la fecha de inicio.
- **Ejemplo:**

  Si el usuario escribe una fecha menor que la de inicio:

  → *"La fecha de fin no puede ser anterior a la fecha de inicio."*

⚠️ Manejo de errores con `try/catch`

- `NumberFormatException`

  Ocurre si el usuario ingresa texto donde se espera un número.

  Ejemplo:

  Ingresar `"dos"` en lugar de `2` como estado → se muestra un mensaje de error sin cerrar el programa.

- `DateTimeParseException`

  Se lanza cuando la fecha tiene un formato incorrecto.

  Ejemplo:

  Ingresar `"mañana"` o `"05-12-2025"` → se informa el error y se solicita la fecha nuevamente.

💡 Resultado

Gracias a estas validaciones, el sistema:

- Previene errores de usuario antes de guardar datos.
- No se rompe ante entradas inválidas.
- Mejora la experiencia, guiando al usuario en tiempo real.
- Asegura que las tareas creadas sean siempre válidas y consistentes.