Informe de resolucion de Trabajo Practico N°3 - Implementación de Tabla Dispersa

Alumno: Tula E. Fernando

Clases:

App

- Definicion de variables para las pruebas e invocacion de constructor de tabla dispersa para obtener un objeto de tabla dispersa que va a contener un arreglo de 101 posiciones.

```
String nombre;
String descripcion;
LocalDateTime fechaFin;
TablaDispersa tabla = new TablaDispersa();
```

- Pruebas:

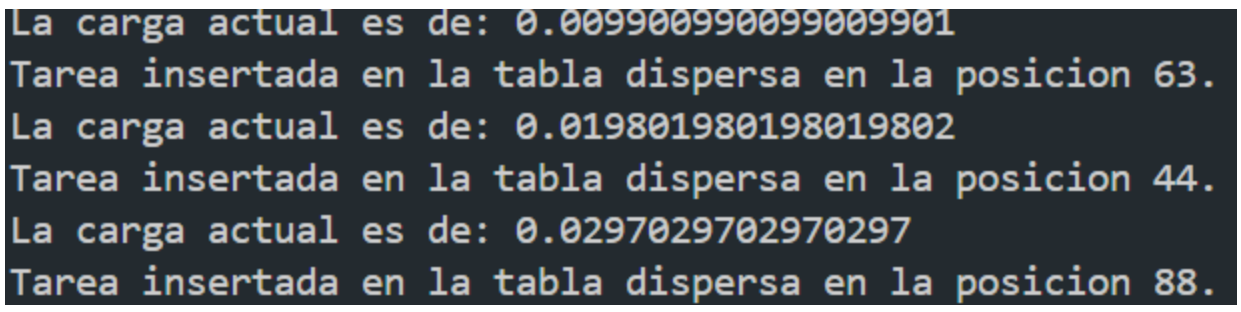
```
nombre = "Tarea 1";
descripcion = "Descripcion de la tarea 1";
fechaFin = LocalDateTime.of(2023, 10, 31, 23, 59); // Fecha de fin de la tarea
```

el metodo insertarTarea llama al contructor de tarea y le pasa los datos y despues los inserta mediante el metodo dispersion en la TablaDispersa, recordar que si bien la tabla tiene otros atributos, varios se crean de forma predeterminada, como la fecha de inicio, el estado, si esta dado de alta y el id.

▼ Insertar tarea

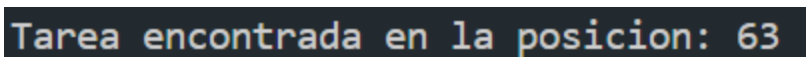
```
tabla.insertarTarea(nombre, descripcion, fechaFin);
```

Cuando intento insertar 3 tareas obtener este resultado en la consola, recordar que a tabla dispersa le solicito en cada insercion el tamaño de la carga y en que posicion se inserto la tarea



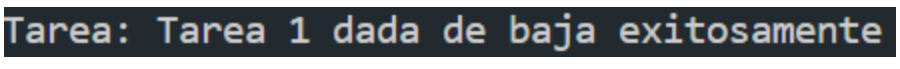
▼ Buscar tarea

```
tabla.buscarTarea("Tarea 1");
```



▼ Dar de baja tarea

```
tabla.bajaTarea("Tarea 1");
```




```
//pruebo buscando la tarea que di de baja
```

```
tabla.buscarTarea("Tarea 1");
```

```
Tarea no encontrada o dada de baja
```

▼ Mostrar tareas: llamo al metodo toString de forma automatica cuando le solicito ver los objetos, recordar que el metodo toString estaba sobreescrito

```
tabla.mostrarInsertados();
```

```
Tarea{id='e7cb2784-9ccb-4bac-939a-6a7c1b76e066', nombre='Tarea 2', descripcion='Descripcion de la tarea 2
', estado=PENDIENTE, fechaInicio=2025-05-10T11:51:56.775464800, fechaFin=2023-11-15T23:59, esAlta=true}

Tarea{id='3b91ad5a-b02e-406c-8d5f-fffbd1222a4c', nombre='Tarea 1', descripcion='Descripcion de la tarea 1
', estado=PENDIENTE, fechaInicio=2025-05-10T11:51:56.772025500, fechaFin=2023-10-31T23:59, esAlta=true}
Tarea{id='5262af19-a577-4f32-86e9-e942b108770a', nombre='Tarea 3', descripcion='Descripcion de la tarea 3
', estado=PENDIENTE, fechaInicio=2025-05-10T11:51:56.775464800, fechaFin=2023-12-01T23:59, esAlta=true}
```

TablaDispersa

Atributos

```
    static final int TAMTABLA = 101;
private Tarea[] tabla = new Tarea[TAMTABLA];
private int cantidadTareas = 0;
private double cargaMaxima = 0.7;
```

Metodos

▼ Constructor: Los atributos del objeto TablaDispersa estan inicializados en el constructor por defecto, pues ya tienen valores asignados al momento de la creación del objeto.

```
public TablaDispersa() {
}
```

▼ Insertar tarea en la tabla de dispersion

```
public void insertarTarea(String nombre, String descripcion, LocalDateTime fechaFin) {
        Tarea tarea = new Tarea(nombre, descripcion, fechaFin);
        //creo un metodo para transformar el id en un entero y despues obtener la posicion del indice
        int direccion = obtenerDireccion(tarea.getId());
        //una vez que la direccion esta disponible se inserta la tarea en la direccion proporcionada por el
        //metodo obtener direccion y aumento la cantidad de tareas para despues hacer el calculo del factor
        //de carga
        tabla[direccion] = tarea;
        cantidadTareas++;
        calcularCarga();
        System.out.println("Tarea insertada en la tabla dispersa en la posicion "+ direccion +".");

    }
```

▼ Obtener direccion: dentro de este metodo se hashea el id de la tarea y tambien se explora si no esta ocupada la direccion donde se desea insertar la tarea

```
public int obtenerDireccion (String clave) {
        double c = 0.6180339887; // Constante para el método de multiplicación
        int intento=0;
        //llamo al metodo transformarString para transformar a un numero double
        //que pueda ser utilizado para obtener posiciones
        double numero = transformarString(clave);

        //aca tengo que obtener el producto entre el numero y la constante
```

```
        double producto = numero * c;

        //aca obtengo la parte decimal del producto
        double decimal = producto - Math.floor(producto);
        //ahora multiplico el decimal por el tamaño de la tabla y hago un casteo
        //para obtener un valor entero
        int posicion = (int)(decimal * TAMTABLA);

        //Llamo a la funcion de exploracion hasta obtener un valor de indice que
        //no este ocupado
        while (tabla[posicion] != null) {
            intento++;
            posicion= resolverColision(posicion, intento);
        }
        return posicion;
    }
```

▼ Transformar cadena de caracteres a id

```
public double transformarString(String clave) {
        long numero = 0;
        //Esto recorre los primeros 10 caracteres del String id
        for (int i=0; i< Math.min(10, clave.length()); i++) {
            numero = numero * 27+ (int) clave.charAt(i); // combina caracteres en base 27
        }
        //aca si bien la variable numero es un long y la funcion retorna un double la funcion aca va a hacer u
        //automatico y no va a perder precision, va a hacer un cast implicito y va a retornar un double
        return numero;
    }
```

▼ Resolver colision en caso de obtener una direccion ya ocupada: El metodo de resolucion de colisiones consiste en una exploracion cuadratica la cual se va incrementando con cada llamada a la funcion para ir obteniendo valores diferentes, hasta que se encuentre el valor de un indice no ocupado en la tabla de dispersion

```
public int resolverColision(int posicion, int intento) {
        return (posicion + intento * intento) % TAMTABLA;
    }
```

▼ Calcular carga de la tabla de dispersion: La funcion devuelve con cada insercion de tareas el porcentaje de ocupacion de posiciones de la tabla de dispersion, de modo que cuando supera la carga maxima que es de 0,7 sale la advertencia de carga maxima superada.

```
public void calcularCarga() {
    double carga= (double)cantidadTareas/TAMTABLA;
    if (carga > cargaMaxima) {
        System.out.println("Se ha superado la carga maxima\n");
    } else {
        System.out.println("La carga actual es de: "+ carga);
    }
}
```

▼ Baja de tarea

```
public void bajaTarea(String nombreTarea) {
    for (int i=0; i<TAMTABLA; i++) {
        if(tabla[i] != null && tabla[i].getNombre() == nombreTarea) {
            tabla[i].setEsAlta(false);
            System.out.println("Tarea: " + nombreTarea +" dada de baja exitosamente\n");
```

```
            cantidadTareas--;
        }
    }
}
```

▼ Mostrar tareas insertadas

```
public void mostrarInsertados() {
        for (int i = 0; i < TAMTABLA; i++) {
            if (tabla[i] != null) {
                //Cuando quiero imprimir un objeto instanciado de una clase,
                //automaticamente llama al metodo toString de esa clase, de modo que
                //no es imprescindible que coloque el metodo que estoy intentando
                //llamar para obtener la informacion.
                System.out.println(tabla[i]);
            }
        }
    }
```

▼ Buscar tarea: Como el arreglo de objetos que estoy usando almacena en sus posiciones punteros a la posicion de memoria donde se almacena cada objeto, cuando las posicones estan vacias no se puede acceder al metodo de un objeto que aun no se ha creado, de modo que primero debo verificar en el condicional si la posicion si no esta vacia y despues verificar si el valor que busco comparar es el mismo y tambien si no la tarea esta de alta

```
public void buscarTarea(String nombreTarea) {
    int encontrado=0;
    for (int i=0; i<TAMTABLA; i++) {
        if(tabla[i] != null && tabla[i].getNombre() == nombreTarea && tabla[i].isEsAlta() == true) {
            System.out.println("\nTarea encontrada en la posicion: "+ i);
            return;
        }
    }
    if (encontrado==0) {
            System.out.println("\nTarea no encontrada o dada de baja");
        }
}
```

Tarea

Atributos

```
private String id;
    private String nombre;
    private String descripcion;
    private Estado estado;
    private LocalDateTime fechaInicio;
    private LocalDateTime fechaFin;
    private boolean esAlta;

    public enum Estado {
        PENDIENTE,
        EN_PROGRESO,
        FINALIZADA
    }
```

Metodos

▼ Constructor: este cosntructor invoca al metodo randomUUID para obtener claves de cardenas de caracteres aleatorias que serviran tanto para identificar la tarea como para hashearla y obtener las posiciones en la tabla de

dispersion

```
public Tarea(String nombre, String descripcion, LocalDateTime fechaFin) {
        this.id = UUID.randomUUID().toString(); // Genera un ID único
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.estado = Estado.PENDIENTE;
        this.fechaInicio = LocalDateTime.now(); // Fecha de inicio es ahora
        this.fechaFin = fechaFin;
        this.esAlta = true; // Por defecto es alta
    }
```

▼ Getters y setters

```
        public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getDescripcion() {
        return descripcion;
    }
    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }
    public Estado getEstado() {
        return estado;
    }

    //metodo que recibe un String y lo asigna al atributo estado y lanza una excepción
    //si el String no es válido, porque recordar que es un dato de tipo enum
    public void setEstado(String estado) {
        try {
            this.estado = Estado.valueOf(estado.toUpperCase());
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("El valor proporcionado no es válido para el atributo Estado. Valores
        }
    }

    //metodo que recibe un objeto Estado y lo asigna al atributo estado y lanza una
    //excepción si el objeto es nulo
    public void setEstado(Estado estado) {
        if (estado == null) {
            throw new IllegalArgumentException("El estado no puede ser nulo.");
        }
        this.estado = estado;
    }
    public LocalDateTime getFechaInicio() {
        return fechaInicio;
    }
    public void setFechaInicio(LocalDateTime fechaInicio) {
        this.fechaInicio = fechaInicio;
```

```
    }
    public LocalDateTime getFechaFin() {
        return fechaFin;
    }
    public void setFechaFin(LocalDateTime fechaFin) {
        this.fechaFin = fechaFin;
    }
    public boolean isEsAlta() {
        return esAlta;
    }
    public void setEsAlta(boolean esAlta) {
        this.esAlta = esAlta;
    }
```

▼ Metodo sobreescrito

```
@Override
    public String toString() {
        return "Tarea{" +
                "id='" + id + '\'' +
                ", nombre='" + nombre + '\'' +
                ", descripcion='" + descripcion + '\'' +
                ", estado=" + estado +
                ", fechaInicio=" + fechaInicio +
                ", fechaFin=" + fechaFin +
                ", esAlta=" + esAlta +
                '}';
    }
```