**FASE 1: IDENTIFICACIÓN DEL PROBLEMA**

**Definición de contexto:**

El proyecto consiste en el desarrollo de un sistema de gestión de tareas y recordatorios. El objetivo principal es permitir a los usuarios agregar, organizar y administrar sus tareas pendientes y recordatorios de manera eficiente y efectiva. Este sistema se basará en el uso de una tabla hash para el almacenamiento de datos, una interfaz de usuario amigable que incluye funciones de agregar, modificar y eliminar tareas, así como la capacidad de ver una lista de tareas y recordatorios ordenados por fecha límite o prioridad. Además, se incorpora una función de gestión de prioridades que distingue entre tareas prioritarias y no prioritarias, utilizando una cola de prioridades para la administración de tareas importantes y un enfoque FIFO para las tareas menos prioritarias. Finalmente, se implementa una función de “deshacer” que utiliza una pila LIFO, para permitir a los usuarios revertir acciones previas en el sistema. Este contexto proporciona una base clara para el diseño y el desarrollo del sistema de gestión de tareas y recordatorios.

**Desarrollo de solución**

Para resolver la situación anterior se eligió el Método de la Ingeniería para desarrollar la solución siguiendo un enfoque sistemático y acorde con la situación problemática planteada. Con base en la descripción del Método de la Ingeniería del libro “Introduction to Engineering” de Paul Wright, se definió el siguiente diagrama de flujo, cuyos pasos seguiremos en el desarrollo de la solución.

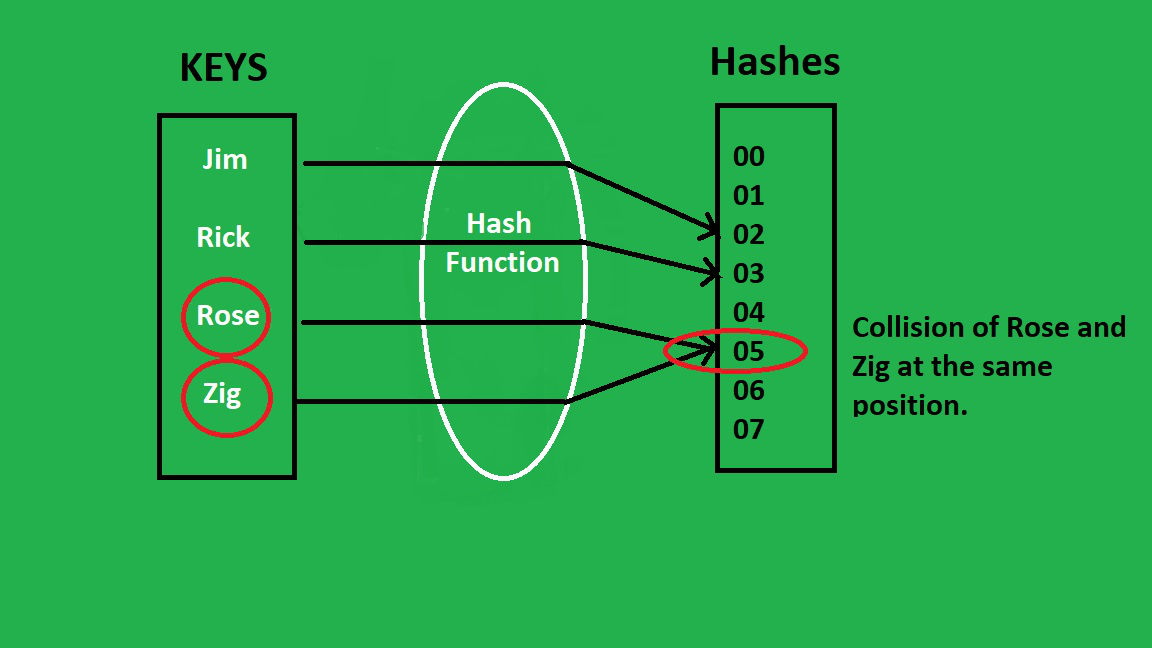
| **Symptoms and needs** | **Description** |
| --- | --- |
| **Creación de tareas y recordatorios** | * Diseñar una función que permita a los usuarios crear nuevas tareas o recordatorios. * Utilizar una tabla hash para almacenar estas tareas y recordatorios, con un identificador único como clave y la información como valor |
| **Interfaz de Usuario Amigable:** | * Desarrollar una interfaz de usuario intuitiva con JavaFx, Swing o en la terminal que permita a los usuarios agregar, modificar y eliminar tareas o recordatorios de forma sencilla. * Proporcionar |
| **Gestión de prioridades de las tareas.** | * Implementar un sistema de priorización que distinga entre las tareas prioritarias y no prioritarias. * Utilizar una cola de prioridades para organizar las tareas prioritarias de acuerdo a su nivel de importancia. * Para las tareas no prioritarias, asegurarse de que manejan en orden de llegada (FIFO). |
| **Función de deshacer a través de las estructuras de Colas** | * Crear una estructura de cola para rastrear las acciones realizadas por los usuarios. * Cada vez que se realice una acción (agregar, modificar o eliminar una tarea), registra la acción y los detalles relacionados en la cola. |
| **Ordenación de Tareas y Recordatorios:** | * Proporcionar la función de ordenar la lista de tareas y recordatorios por fecha límite o prioridad en la interfaz de usuario. |
| **Almacenamiento eficiente de datos** | * El software debe tener un diseño e implementación de un sistema de almacenamiento eficiente para las tareas y recordatorios en la tabla hash. |
| **Documentación** | * Proporcionar documentación clara y recursos de capacitación para que los usuarios comprendan cómo utilizar las funciones del sistema de gestión de tareas y recordatorios. |
| **Pruebas y depuración** | * El proyecto se debe realizar pruebas del sistema para identificar y corregir posibles errores o problemas de funcionamiento. |
| **Optimización de rendimiento** | * Optimizar el rendimiento del sistema para que sea rápido y eficiente cuando se trabaja con grandes cantidades de tareas y recordatorios. |

* El sistema debe permitir crear tareas o recordatorios (en una tabla hash).
* La interfaz debe permitir al usuario: crear, modificar y eliminar recordatorios o tareas.
* El software debe permitir gestionar las prioridades de las tareas: Hacer primero las tareas de tipo prioritarias y las no prioritarias hacerlas en orden de llegada (FIFO).
* Permitir al usuario, mediante la interfaz, deshacer la última acción hecha mediante el uso de la estructura de colas.

**FASE 2: RECOPILACIÓN DE LA INFORMACIÓN NECESARIA**

Estructuras a usar:

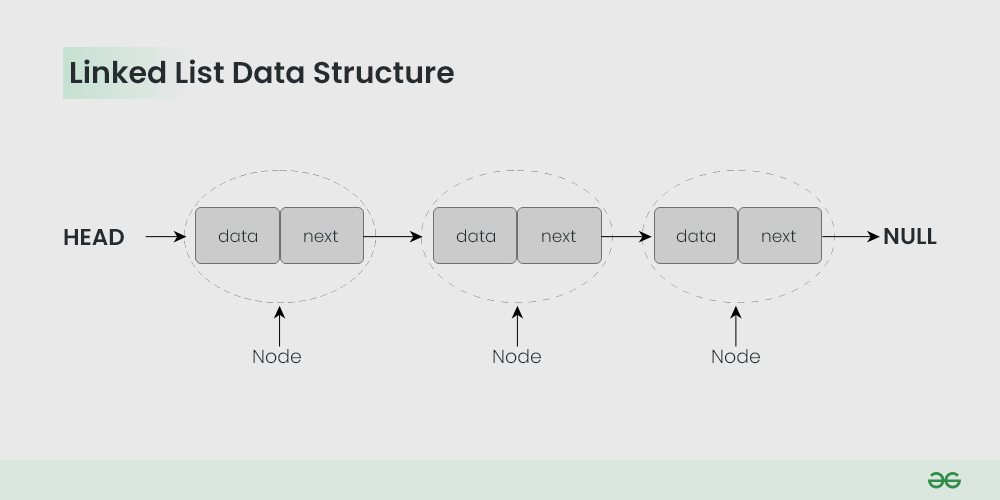
* Tabla hash: Son estructuras de datos que se basan en asociar una clave o llave a un valor dentro de la tabla hash, para esto se usa una función hash que convierte la clave en un valor numérico que es la llave, posición dentro de la tabla hash del valor. Esta estructura es muy eficiente en búsqueda pues tiene una complejidad O(1), es decir, una complejidad que no varía según la cantidad de elementos. Para no reservar demasiados espacios de memoria, dentro de la función hash, se usa el operador módulo limitando la cantidad de posibles resultados. Para solucionar colisiones, cuando varios valores se guardan en la misma llave, se usan listas enlazadas convirtiendo esa posición en la cabeza de esta lista.



GeeksforGeeks. (s.f.). Java Program to Implement Hash Tables Chaining with Doubly Linked Lists. Recuperado de: <https://www.geeksforgeeks.org/java-program-to-implement-hash-tables-chaining-with-doubly-linked-lists/>

Oracle. (2014). Class Hashtable. Recuperado de <https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html>

* Lista enlazada: Es una estructura de datos lineal en la que cada uno de los datos tiene una referencia al siguiente, así solo se tiene que tener referencia al primer elemento de la lista, la cabeza, para guardar toda la lista.



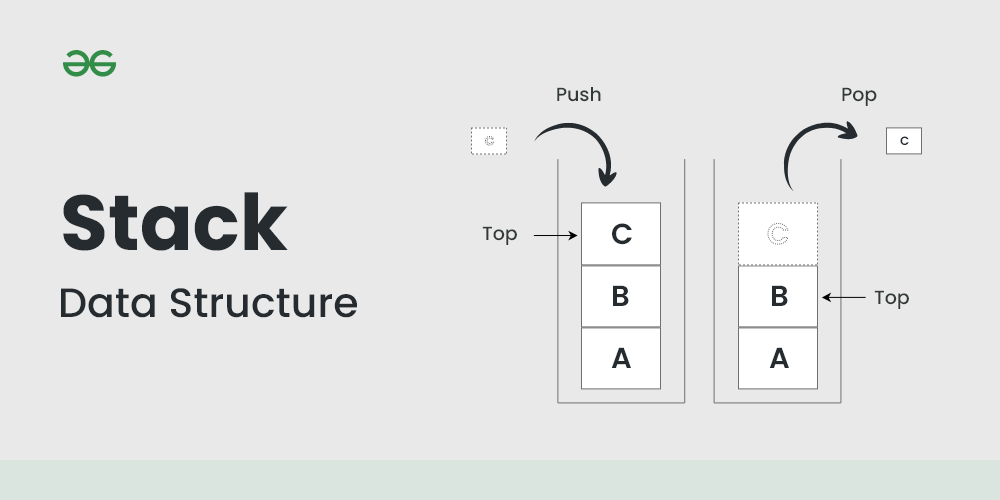
GeeksforGeeks. (s.f.). Linked List. Recuperado de https://www.geeksforgeeks.org/data-structures/linked-list/

* Lista doblemente enlazada: Es una variación de la estructura de lista enlazada, la única diferencia es que, aparte de tener referencia al siguiente, todos los datos tienen una referencia al dato anterior.

GeeksforGeeks. (s.f.). Introduction and Insertion in a Doubly Linked List. Recuperado de <https://www.geeksforgeeks.org/introduction-and-insertion-in-a-doubly-linked-list/>

Oracle. (2014). Class LinkedList. Recuperado de <https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html>

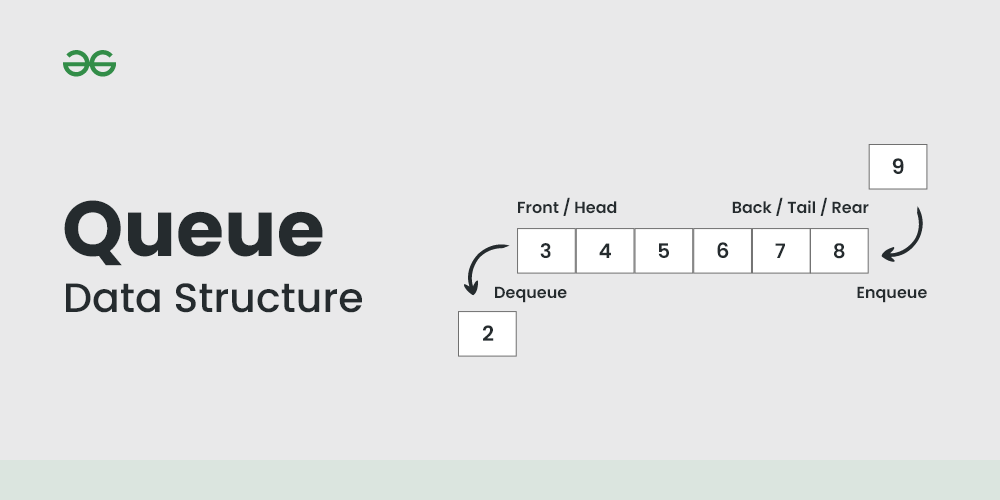
* Pilas: Estructura de datos en la que solo se tiene referencia al último elemento añadido, usa el principio de LIFO (“last in, first out” o ultimo en entrar primero en salir). Una forma de entenderlo es una pila de platos, solo se ponen platos uno encima del último, al igual que solo se saca el último plato en la pila.



GeeksforGeeks. (s.f.). Stack Data Structure. Recuperado de <https://www.geeksforgeeks.org/stack-data-structure/?ref=lbp>

Oracle. (2014). Class Stack. Recuperado de <https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html>

* Colas: Son un tipo de estructura en la cual solo se puede eliminar el último elemento de la cola y que solo se pueden añadir datos al principio de la cola, cumple el principio de FIFO (“first in, first out” o primero en entrar, primero en salir). Un ejemplo bien podría ser una fila de cualquier local o servicio, asumiendo que las personas no se colen, siempre el primero en llegar será el primero en salir.



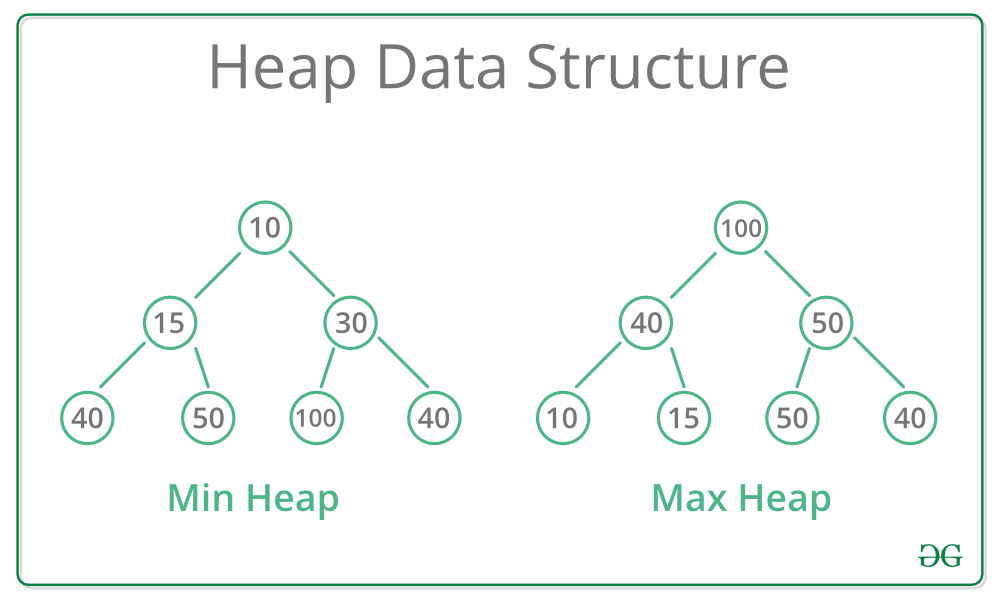
GeeksforGeeks. (s.f.). Queue Data Structure. Recuperado de <https://www.geeksforgeeks.org/queue-data-structure/?ref=lbp>

Oracle. (2014). Interface Queue. Recuperado de <https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html>

* Colas de prioridad: Es un tipo especial de cola en la que se agrupan los datos según su valor de prioridad haciendo que los valores con mayor prioridad salgan primero, en caso de que dos valores tengan la misma prioridad saldrá primero el que entró primero. Un ejemplo de esto sería la fila de atención de un hospital, se le da prioridad a enfermedades más graves, a mujeres embarazadas, niños y personas de la tercera edad; en este ejemplo se puede ver la complejidad con la que dar la prioridad a las personas si hay varias personas con características diferentes o peor aún si hay personas con varias de estas características.

GeeksforGeeks. (s.f.). Priority Queue - Introduction. Recuperado de <https://www.geeksforgeeks.org/priority-queue-set-1-introduction/>

Oracle. (2014). Class PriorityQueue. Recuperado de <https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html>

* Heaps: Es una estructura de datos que se modela como un árbol binario pero, al contrario de un BST que guarda referencias a los hijos, este puede guardarse en un arraylist o un array común. Este puede ser un Max Heap que el padre siempre es mayor a sus hijos o un Min Heap en el cual el padre siempre es menor a sus hijos. Los elementos que guarda el heap deben ser comparables. Existen varias funciones de esta estructura: Heapify (crea un heap dado un array 0 (log n)), insertion (inserta un valor en el heap), eliminación (Elimina la raiz y acomoda nuevamente el heap), etc

GeeksforGeeks. (s.f.). Heap Data Structure. GeeksforGeeks. Recuperado de <https://www.geeksforgeeks.org/heap-data-structure/>

Oracle. (2014). Class PriorityQueue. Recuperado de <https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html>

**FASE 3: BÚSQUEDA DE SOLUCIONES CREATIVAS**

través de una lluvia de ideas se encontraron las siguientes soluciones para cada uno de los requerimientos del sistema

**Almacenar tareas y recordatorios**

Double linked list

Hash Table

Arraylist

array

·  **Interfaz de usuario**

AWT

Swing

Consola

JavaFX

· **Gestion tareas prioritarias**

Priority Queue

Queue

Binary Search Tree

Ordered Linked List

· **Gestión no prioritarias**

Priority Queue

Queue

Binary Search Tree

Ordered Linked List

·  **Metodo deshacer**

o Stack

o Accion register + Stack

o Changes register

**FASE 4: TRANSICIÓN DE LA FORMULACIÓN DE IDEAS A LOS DISEÑOS PRELIMINARES**

**A. Almacenar tareas y recordatorios**:

En el almacenamiento de tareas y recordatorios se escogió finalmente una Hash Table, debido a que la double linkedList y los arrays no nos permitía tener una unión tan clara y fija del identificador con su información a diferencia que la tabla hash ya tenía implementado esta forma de unir las dos secciones de clave y valor (identificador e información).

**b. Interfaz de Usuario:**

Decidimos que JavaFX sera la librería que nos ayudará a crear la interfaz gráfica que necesitamos puesto a que la consola no nos da los bonus útiles para nuestra implementación, swing tiene un uso demasiado básico y AWT no es una forma de interfaces gráficas que trabajaremos mucho en el futuro, por ende, JavaFX es más fácil de utilizar y mejor de utilizar en nuestro contexto..

**c. Gestión de Prioridades:**

La gestión de tareas prioritarias al necesitar organizar las tareas según su nivel de importancia, necesitamos que se ajuste a su nivel de importancia, por ende, para manejar primero las más importantes las otras estructuras, aunque pueden llegar a funcionar nos quedamos con Priority Queue por su estructura ideal para atender según la necesidad.

La gestión de tareas no prioritarias al solo tener que manejar el orden de llegada FIFO, solo es necesario utilizar un Queue que nos permita tomar las tareas en el orden de llegada que haya pasado, siendo así que el resto de estructuras serian peores de utilizar para este caso.

**d. Método para Deshacer:**

Implementar una pila (LIFO) para realizar un seguimiento de las acciones realizadas por el usuario en el sistema.

Registrar cada acción realizada por el usuario, incluyendo detalles como la acción específica ("Agregar tarea", "Modificar tarea", "Eliminar tarea") y los detalles de la tarea afectada.

Para realizar el método de deshacer podemos utilizar un Stack que contenga un seguimiento de las acciones del usuario, creando y registrando las acciones hasta que se llame al método deshacer que quita la última acción del usuario, al contener el stack el historial de cambios no es necesario ningún register extra para solucionar el requerimiento.

**f. Uso del Método de Deshacer:**

En la interfaz de usuario, proporcionar a los usuarios la opción de deshacer la última acción realizada. Cuando el usuario selecciona la opción "Deshacer", se debe llamar al método deshacer(), lo que revertirá la última acción realizada.

Cada una de estas soluciones aborda una parte específica del problema y contribuye al diseño y desarrollo del sistema de gestión de tareas y recordatorios. Es importante tener en cuenta que estas soluciones pueden complementarse y adaptarse según las necesidades y requisitos específicos del proyecto.

**Fase 5: Evaluation and Selection of the Best Solution:**

Finalmente, en este escrito se realizará la selección de la mejor solución. Para ello se establecieron una serie de criterios que se calificarán del 1 al 5 (donde 5 es el puntaje más alto considerado excelente y 1 es el puntaje más bajo considerado regular) y permitirán elegir la mejor propuesta de solución. Los criterios son los siguientes:

- Efficiency: Which of the proposed solutions is more efficient in terms of time and resources?

-Maintainability:

- Scalability: Can the solution handle large amounts of data and users? Can it adapt to future changes in the number of users and data?

- Usability: Is the solution easy to use for end-users? Is it easy to implement.

**Almacenar tareas y recordatorios**

| Solutions\evaluation | Efficiency | Maintainability | Scalability | Usability | Total |
| --- | --- | --- | --- | --- | --- |
| Double linked list | 2 | 5 | 2 | 2 | 11 |
| Hash Table | 4 | 5 | 5 | 4 | 18 |
| Arraylist | 3 | 5 | 5 | 4 | 17 |
| Array | 4 | 1 | 1 | 1 | 8 |

**Interfaz de usuario**

| Solutions\evaluation | Efficiency | Maintainability | Scalability | Usability | Total |
| --- | --- | --- | --- | --- | --- |
| AWT | 2 | 2 | 2 | 2 | 8 |
| Swing | 3 | 2 | 2 | 3 | 10 |
| Consola | 1 | 2 | 2 | 2 | 7 |
| JavaFX | 5 | 5 | 5 | 4 | 19 |

**Gestión tareas prioritarias**

| Solutions\evaluation | Efficiency | Maintainability | Scalability | Usability | Total |
| --- | --- | --- | --- | --- | --- |
| Priority Queue | 5 | 5 | 5 | 3 | 18 |
| Queue | 4 | 3 | 3 | 4 | 14 |
| BST | 3 | 3 | 3 | 2 | 11 |
| Ordered Linked list | 1 | 5 | 1 | 5 | 12 |

**Gestión no prioritarias**

| Solutions\evaluation | Efficiency | Maintainability | Scalability | Usability | Total |
| --- | --- | --- | --- | --- | --- |
| Priority Queue | 4 | 4 | 5 | 3 | 16 |
| Queue | 5 | 5 | 5 | 4 | 19 |
| BST | 2 | 2 | 2 | 2 | 8 |
| Ordered Linked list | 4 | 4 | 4 | 5 | 17 |

**Método deshacer**

| Solutions\evaluation | Efficiency | Maintainability | Scalability | Usability | Total |
| --- | --- | --- | --- | --- | --- |
| Stack | 5 | 5 | 5 | 4 | 19 |
| Stack + acción register | 5 | 5 | 5 | 5 | 20 |
| changes register | 4 | 4 | 4 | 4 | 16 |

Por las evaluaciones anteriores se llegó a la conclusión que las mejores soluciones son: Hash table para almacenar tareas y recordatorios, para la interfaz javaFX, priority queue para las tareas prioritarias, queue para no prioritarias y para deshacer la última acción Stack + accion register. Todos fueron escogidos por tener la suma de puntajes más alta en su respectivo problema además de que algunos también representan de mejor manera el problema.