Pablo Andres Guzman Alarcon A00399523
Oscar Stiven Muñoz Ramirez A00399922
Diego Armando Polanco Lozano A00399926
Computación Y Estructuras Discretas l

Engineering method

PHASE 1: PROBLEM IDENTIFICATION

context:

The project consists of the development of a task and reminder management system. The main goal is to allow users to add, organize and manage their to-dos and reminders efficiently and effectively. This system will be based on the use of a hash table for data storage, a friendly user interface that includes add, modify and delete tasks functions, as well as the ability to view a list of tasks and reminders sorted by priority. Additionally, a priority management function is incorporated that distinguishes between priority and non-priority tasks, using a priority queue for the management of important tasks and a FIFO approach for lower priority tasks. Finally, an "undo" function that uses a LIFO stack is implemented to allow users to revert previous actions in the system. This context provides a clear basis for the design and development of the task and reminder management system.

Development solution

To resolve the previous situation, the Engineering Method was chosen to develop the solution following a systematic approach and in accordance with the problematic situation posed. Based on the description of the Engineering Method from the book "Introduction to Engineering" by Paul Wright, the following flow diagram was defined, whose steps we will follow in the development of the solution.

Symptoms and needs	**Description**
Creating tasks and reminders	- Design a feature that allows users to create new tasks or reminders. - Use a hash table to store these tasks and reminders, with a unique identifier as the key and the information as the value
Friendly User Interface:	- Develop an intuitive user interface with JavaFx, Swing or in the terminal that allows users to easily add, modify and delete tasks or reminders. - Provide
Task priority management.	- Implement a prioritization system that distinguishes between priority and non-priority tasks. - Use a priority queue to organize priority tasks according to their level of importance. - For non-priority tasks, make sure they handle them on a first-in, first-out(FIFO)

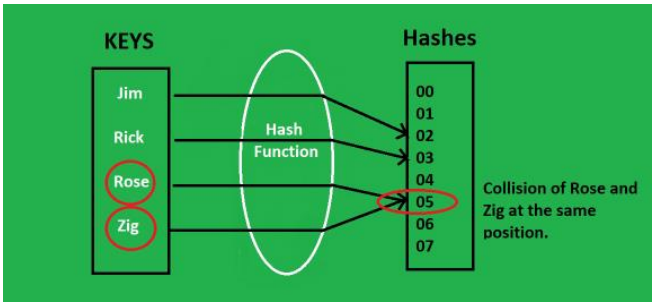
	basis.
Undo function through Queue structures	- Create a queue structure to track actions performed by users. - Every time an action is performed (add, modify, or delete a task), log the action and related details in the queue.
Ordering of Tasks and Reminders:	- Provide the feature of sorting the list of tasks and reminders by deadline or priority in the user interface.
Efficient data storage	- The software must have a design and implementation of an efficient storage system for tasks and reminders in the hash table.
Documentation	- Provide clear documentation and training resources so that users understand how to use the functions of the task and reminder management system.
Testing and debugging	- The project must perform system testing to identify and correct possible errors or operational problems.
Performance optimization	- Optimize system performance to be fast and efficient when working with large amounts of tasks and reminders.

- The system must allow you to create tasks or reminders (in a hash table).
- The interface must allow to: create, modify and delete reminders or tasks.
- The software must manage tasks by priorities: Do priority type tasks first, and do non-priority tasks in order of arrival (FIFO).
- Allow the user, through the interface, to undo the last action done by using the queue structure.

PHASE 2: COLLECTION OF THE NECESSARY INFORMATION

Structures to use:

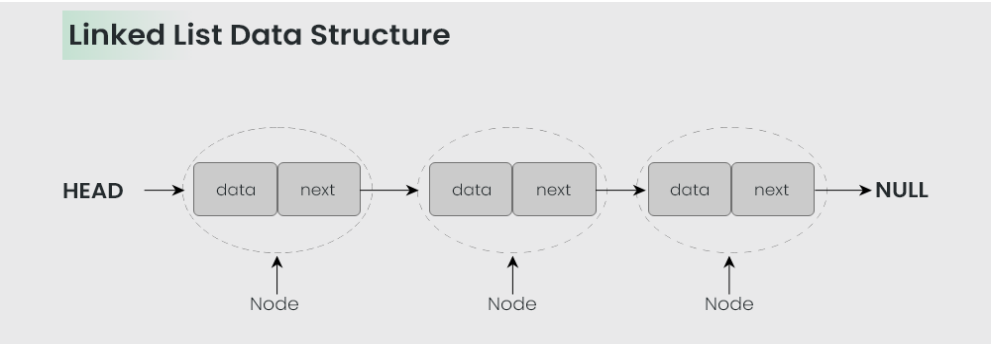
- Hash table: These are data structures that are based on associating a key with a value within the hash table. For this, a hash function is used that converts the key into a numerical value that is the position within the table. hash of the value. This structure is very efficient in search because it has an O(1) complexity, that is, a complexity that does not vary depending on the number of elements. In order not to reserve too many memory spaces, within the hash function, the mod operator (in java %) is used, limiting the number of possible results. To resolve collisions, when several values are stored in the same position, linked lists are used by making that position the head of this list.



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. Recovered fromhttps://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html

- Linked list: It is a linear data structure in which each of the elements has a reference to the next, so you only have to have a reference to the first element of the list, the head, to save the entire list.



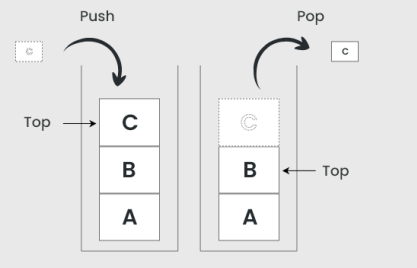
GeeksforGeeks. (s.f.). Linked List. Recuperado de https://www.geeksforgeeks.org/data-structures/linked-list/

- Doubly linked list: It is a variation of the linked list structure, the only difference is that, apart from having a reference to the next one, all the data has a reference to the previous element.

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

Oracle. (2014). Class LinkedList. Recovered from https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html

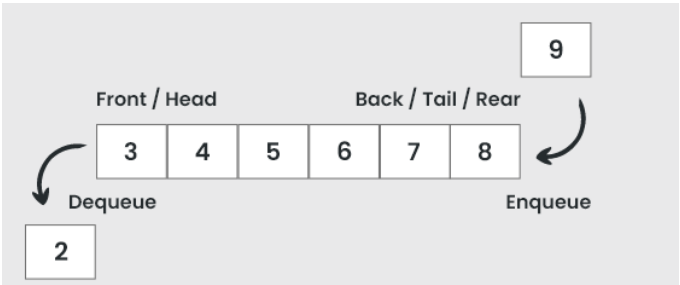
- Stacks: Data structure in which only the last added element is referenced, uses the LIFO principle (“last in, first out”or last in first out). One way to understand it is a stack of plates, you only put one on top of the last, just like you only take out the last plate in the stack.



GeeksforGeeks. (s.f.). Stack Data Structure. Recovered from https://www.geeksforgeeks.org/stack-data-structure/?ref=lbp

Oracle. (2014). ClassStack. Recovered from https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html

- Queues: They are a type of structure in which only the last element of the queue can be removed and data can only be added to the beginning of the queue, complying with the FIFO principle ("first in, first out" or first in , first out). An example could well be a line at any store or service, normally, the first to arrive will always be the first to leave.



GeeksforGeeks. (s.f.). Queue Data Structure. Recovered from https://www.geeksforgeeks.org/queue-data-structure/?ref=lbp

Oracle. (2014). InterfaceQueue. Recovered from https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html

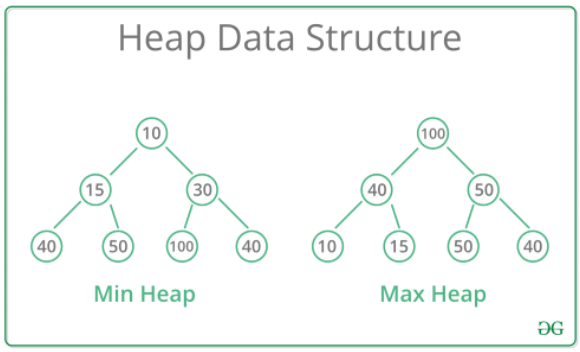
- Priority queues: It is a special type of queue in which data is grouped according to its priority value, making the values with the highest priority come out first. If two values have the same priority, the one that entered first will come out first. An example of this would be the queue for care in a hospital, priority is given to more serious illnesses, pregnant women, children and the elderly; In this example you can see the complexity how to distribute priority to people if there are several people with different characteristics or worse if there are people with several of these characteristics.

GeeksforGeeks. (s.f.). Priority Queue - Introduction. Recovered from https://www.geeksforgeeks.org/priority-queue-set-1-introduction/

Oracle. (2014). Class PriorityQueue. Recovered from https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html

- Heaps: It is a data structure that is modeled as a binary tree, but it can also be stored in an arraylist or a common array. This can be a Max Heap in which the father is always bigger than his children or a Min Heap in which the father is always smaller than his children. The elements stored in the heap must be comparable. There are several

functions of this structure: Heapify (creates a heap given a 0 (log n) array), insertion (inserts a value into the heap), elimination (Deletes the root and rearranges the heap), etc.



GeeksforGeeks. (s.f.). Heap Data Structure. GeeksforGeeks. Recovered from https://www.geeksforgeeks.org/heap-data-structure/

Oracle. (2014). Class PriorityQueue. Recovered from https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html

PHASE 3: SEARCH FOR CREATIVE SOLUTIONS

Through brainstorming, the following solutions were found for each of the system requirements.

Store tasks and reminders

Double linked list

Hash Table

Arraylist

array

- **User interface**

AWT

Swing

Console

JavaFX

- **Management of priority tasks**

Priority Queue

Queue

Binary Search Tree

Ordered Linked List

- **Non-priority management**

  Priority Queue

  Queue

  Binary Search Tree

  Ordered Linked List

- **undo method**

  Stack

  Action register + Stack

  Changes register

PHASE 4: TRANSITION FROM IDEA FORMULATION TO PRELIMINARY DESIGNS

A. Store tasks and reminders:

In the storage of tasks and reminders, a Hash Table was finally chosen, because the double linkedList and the arrays did not allow us to have such a clear and fixed union of the identifier with its information, unlike the hash table already had this form of implementation implemented. join the two sections of key and value (identifier and information).

b. User Interface:

We decided that Swing will be the library that will help us create the graphical interface we need since the console does not give us useful bonuses for our implementation, it is too basic but it is so easy to implement, more than javaFX and AWT. the future, therefore, Swing is easier to use and better to use in our context.

c. Priority Management:

The management of priority tasks, when we need to organize the tasks according to their level of importance, we need them to adjust to their level of importance, therefore, to manage the most important ones first, the other structures, although they may work, we stick with Priority Queue for its ideal structure to serve according to need.

The management of non-priority tasks by only having to manage the FIFO order of arrival, it is only necessary to use a Queue that allows us to take the tasks in the order of arrival that has passed, thus the rest of the structures would be worse to use for this case.

d. Undo Method:

Implement a stack (LIFO) to track actions taken by the user in the system.

Record every action performed by the user, including details such as the specific action ("Add Task", "Modify Task", "Delete Task") and the details of the affected task.

To perform the undo method, we can use a Stack that contains a trace of the user's actions, creating and recording the actions until the undo method is called, which removes the last user action.

e. Using the Undo Method:

In the user interface, provide users with the option to undo the last action performed. When the user selects the "Undo" option, the undo() method must be called, which will revert the last action performed.

Each of these solutions addresses a specific part of the problem and contributes to the design and development of the task and reminder management system. It is important to note that these solutions can complement and adapt according to the specific needs and requirements of the project.

Phase 5: Evaluation and Selection of the Best Solution:

Finally, in this writing the selection of the best solution will be made. For this purpose, a series of criteria were established that will be rated from 1 to 5 (where 5 is the highest score considered excellent and 1 is the lowest score considered average) and will allow the best solution proposal to be chosen. The criteria are the following:

- 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.

Store tasks and reminders

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

User interface

Solutions\evaluation	Efficiency	Maintainability	Scalability	Usability	Total
AWT	2	2	2	2	8
Swing	3	2	2	5	12
Console	1	2	2	2	7
JavaFX	3	3	3	1	10

Management of priority tasks

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

Non-priority management

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

undo method

Solutions\evaluation	Efficiency	Maintainability	Scalability	Usability	Total

Stack	5	5	5	4	19
Stack + register action	5	5	5	5	20
changes register	4	4	4	4	16

Based on previous evaluations, it was concluded that the best solutions are: Hash table to store tasks and reminders, for the java swing interface, priority queue for priority tasks, queue for non-priority tasks and to undo the last action, we are going to use a Stack + register structure. All were chosen for having the highest sum of scores in their respective problem, in addition to the fact that some also represent in a better way the problem.

Step 2 Analysis:

#1 Search function of Heap class

```
function search(node)
    search = null                                   1
    for i = 1 to length(heapArr) - 1 do             n
        if heapArr[i].getNode() == node then        1
            search = heapArr[i].getNode()           1
            break                                   1
        end if
    end for
    return search                                   1
```

Type	**Variable**	**Amount atomic values**
Input	node	1
Input	heapArr	n
Auxiliary	i	1
output	search	1

Pablo Andres Guzman Alarcon A00399523
Oscar Stiven Muñoz Ramirez A00399922
Diego Armando Polanco Lozano A00399926
Computación Y Estructuras Discretas l

The time complexity of the **search(node)** function, starting with the lines of code, would be 1 + n + 1 + 1 + 1 + 1 = n + 5. Therefore, in the end, its time complexity would be O(n) because n is the highest-order term of the complexity.

Regarding the space complexity of the **search(node)** function, summing up the atomic values would be 1 + n + 1 + 1 = n + 3. Therefore, in the end, its space complexity would be O(n) because n is the highest-order term of the complexity.

#2 Clone function of Queue class

```
clone_Queue() {
    clone = new Queue()                    1
    current = first                        1

    while current != null do               n
        clone.offer(current.getValue())    1
        current = current.getNext()        1

    return clone                           1
}
```

Type	**Variable**	**Amount atomic values**
Input	first	1
Auxiliary	current	1
Output	clone	n

The time complexity of the `clone_Queue()` function, starting with the lines of code, would be 1 + 1 + n + 1 + 1 + 1 = n + 5. Therefore, in the end, its time complexity would be O(n) because n is the highest-order term of the complexity.

Regarding the space complexity of the `clone_Queue()` function, summing up the atomic values would be 1 + 1 + n = n + 2. Therefore, in the end, its space complexity would be O(n) because n is the highest-order term of the complexity.

Pablo Andres Guzman Alarcon A00399523
Oscar Stiven Muñoz Ramirez A00399922
Diego Armando Polanco Lozano A00399926
Computación Y Estructuras Discretas l