**THE DESIGN METHOD IN ENGINEERING**

**PHASE 1. IDENTIFYING THE PROBLEM**

**Problem identification:**

A group of software engineers called YZ Group wants to build a system that allows the management of tasks and reminders, due to the increasing need to organize daily responsibilities and stay on top of pending tasks in an increasingly busy world. Creating a task and reminder management system seeks to help people be more productive and meet their commitments effectively.

|  |  |
| --- | --- |
| **Customer** | YZ Group |
| **User** | Clients of YZ Group |
| **Functional requirements** | * FR 1. Create a task or reminder. * R.F 2. Deleted a task or reminder. * R.F 3. Modify a task or reminder. * R.F 4. Show the organize a task or reminder by priority. * R.F 5. Save changes made. * R.F 6. Undo changes. |
| **Context of the problem** | A group called YZ Group wants to build a system that allows the management of tasks and reminders, because they want the users to be able to write down in the system those important things that they want to remember. |
| **Non-functional requirements** | NFR1: Scalable Software |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name or identifier** | FR 1. Create a task or reminder | | | |
| **Summary** | The system must allow the creation of tasks or reminder | | | |
| **Inputs** | **input name** | **Data type** | | **Selection or repetition condition** |
| id | String | | Every task or reminder has a unique identification code that is generated by the system |
| tittle | | String | Every task or reminder has a tittle. |
| description | | String | Every task or reminder can have a description without character. |
| dateLim | | Date | Some tasks and reminders have a deadline to change their status |
| priority | | int | Is the level of importance of the task or reminder |
| **General activities needed to obtain the results** | * Receive the attributes of the task or reminder. | | | |
| **Result or postcondition** | The task or reminder was successfully created | | | |
| **Outputs** | **Output name** | | **Data type** | **Selection or repetition condition** |
| errorMessage | String | | Show was an error during the creation of the task or reminder. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name or identifier** | FR 2. Deleted a task or reminder | | | |
| **Summary** | The system must allow the elimination of tasks or reminder previously created. | | | |
| **Inputs** | **input name** | **Data type** | | **Selection or repetition condition** |
| id | String | | The unique identification must be used to search and delete the task or reminder |
| **General activities needed to obtain the results** | * Search the task or reminder to deleted. * Remove the task or reminder of the list of activities. | | | |
| **Result or postcondition** | The task or reminder was successfully deleted | | | |
| **Outputs** | **Output name** | | **Data type** | **Selection or repetition condition** |
| errorMessage | String | | Show it was an error during the elimination of the task or reminder. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name or identifier** | FR 3. Modify a task or reminder | | | |
| **Summary** | The system must allow the modification of tasks or reminder | | | |
| **Inputs** | **input name** | **Data type** | | **Selection or repetition condition** |
| id | String | | The unique identification must be used to search and modify the task or reminder |
| change | | String | The changes that the user wants to do on the select task or reminder |
| **General activities needed to obtain the results** | * Search the task or reminder to modify. * Save the changes made to the task or reminder | | | |
| **Result or postcondition** | The task or reminder selected was successfully modify and save. | | | |
| **Outputs** | **Output name** | | **Data type** | **Selection or repetition condition** |
| errorMessage | String | | Show it was an error during the modifying of the task or reminder. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name or identifier** | FR 4. Show the organize a task or reminder by priority | | | |
| **Summary** | The system must allow the view of the tasks or reminder organize by the priority | | | |
| **Inputs** | **input name** | **Data type** | | **Selection or repetition condition** |
|  |  | |  |
| **General activities needed to obtain the results** | * Get the priority of every element on the list of task or reminder | | | |
| **Result or postcondition** | The tasks or reminders is show in an organized way | | | |
| **Outputs** | **Output name** | | **Data type** | **Selection or repetition condition** |
| msg | String | | Show the list of tasks or reminders organized by priority |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name or identifier** | FR 5. Save changes made | | | |
| **Summary** | The system must save the changes, leaving first the last activity made for the user | | | |
| **Inputs** | **input name** | **Data type** | | **Selection or repetition condition** |
| action | String | | It’s used every time that the user realizes and action |
| **General activities needed to obtain the results** | * Every time that the user realizes an action get a description of what the user did. | | | |
| **Result or postcondition** | The actions made for the user are saved in the system to be used later. | | | |
| **Outputs** | **Output name** | | **Data type** | **Selection or repetition condition** |
|  |  | |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name or identifier** | FR 6. Undo changes | | | |
| **Summary** | The system must undo the last changes made | | | |
| **Inputs** | **input name** | **Data type** | | **Selection or repetition condition** |
| action | String | | It’s the last action made for the user |
| **General activities needed to obtain the results** | * Go to the list of activities made for the user. * Remove/deleted the last activity. | | | |
| **Result or postcondition** | The last action made for the user is deleted. | | | |
| **Outputs** | **Output name** | | **Data type** | **Selection or repetition condition** |
| errorMessage | String | | Show it was an error during the undo of the action. |

**PHASE 2. GATHERING OF THE NECESSARY INFORMATION**

**Import clarification to kept in mind:**

The client wants that the tasks/reminders be displayed in a certain order, in this case, by their priority, at a higher priority level they will be seen first. If the tasks/reminders have the same priority level, they will be displayed according to which one was created first. Since the tasks/reminders will be organized according to their priority and the first item will be the first out, you can see that the display of tasks/reminders follows the same behavior as a structure called a priority queue.

For the deletion of changes, it is known that only the last change made will be deleted, for this it is necessary that the last change made be the first to be deleted in the task/reminder management system, in this way it is known that elimination follows the same behavior as a structure call stack.

Every time that the user wants to modify or deleted a task/reminder, it should be search first, is for that reason that the client wishes to the task/reminder been save in a way that makes it easy and inexpensive for the system to search for them when its needed. Because the programming language asked to be used is Java, the team decided that it would be prudent to use a structure called Hash Table to fulfill the need to optimize task/reminder management.

**Needed information:**

***Hash Table***

A hash table is a data structure that is used to store keys/value pairs. It uses a hash function to compute an index into an array in which an element will be inserted or searched. By using a good hash function, hashing can work well. Under reasonable assumptions, the average time required to search for an element in a hash table is O (1), which means that is not needed to go through the entire structure to find the element.

But what is defined as hashing, well, hashing is a technique that is used to uniquely identify a specific object from a group of similar objects. An example of how hashing is used is in the universities, where each student is assigned a unique roll number.

The idea of hashing is to distribute entries uniformly across an array. Each element is assigned a key (converted key). Using the key, the algorithm (hash function) computes an index that suggests where an element can be found or inserted.

***Possible options of hash:***

**Open hashing:** Data can be stored in the form of chained lists within an infinite space, at least in theory. Although the keys are limited, chaining allows larger amounts of data to be processed.

**Closed hashing:** The number of keys is limited by the capacity of the table. If you try to save more data, a so-called overflow occurs. With each new scan, the table is polled for free positions in which to locate overflowed items.

All these concepts can be seen very abstract, so let's look at the following example:

Assume that you must store strings in the hash table by using the hashing technique {“abcdef”, “bcdefa”, “cdefab”, “defibs”}. To compute the index for storing the strings, use a hash function that states the following:

The index for a specific string will be equal to sum of ASCII values of characters added to the end their respective order in the string after which it is modulo with 2069, the ASCII values of a, b, c, d, e, and f are 97, 98, 99, 100, 101, and 102 respectively and as 20069 is a prime number, it will reduce the possibility of indexing different strings (collisions).

|  |  |  |
| --- | --- | --- |
| String | Hash function | Index |
| abcdef | (971 + 982 + 993 + 1004 + 1015 + 1026) %2069 | 38 |
| bcdefa | (981 + 992 + 1003 + 1014 + 1025 + 976) %2069 | 23 |
| cdefab | (991 + 1002 + 1013 + 1024 + 975 + 986) %2069 | 14 |
| defabc | (1001 + 1012 + 1023 + 974 + 985 + 996) %2069 | 11 |

The result will be the next:



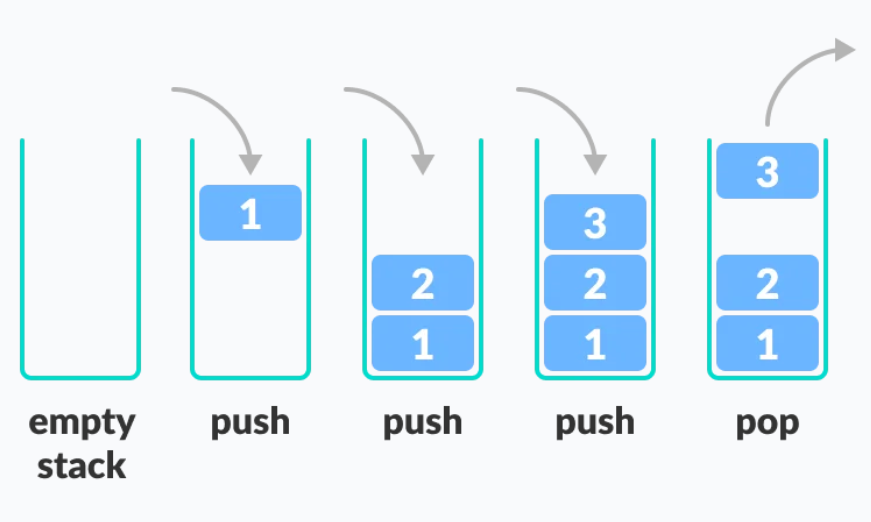
(Hacker Earth, 2016)

***Stack***

A stack is a linear data structure that follows the principle of Last in First Out (LIFO). This means the last element inserted inside the stack is removed first. You can think of the stack data structure as the pile of plates on top of another. Here, you can put a new plate on top or remove the top plate and, if you want the plate at the bottom, you must first remove all the plates on top. This is exactly how the stack data structure works.

There are some basic operations that allow us to perform different actions on a stack.

* Push: Add an element to the top of a stack.
* Pop: Remove an element from the top of a stack.
* Is-Empty: Check if the stack is empty.
* Peek: Get the value of the top element without removing it.



(Programiz, 2023)

***Queue***

A queue is a useful data structure in programming. It is like the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket. Queue follows the First in First Out (FIFO) rule - the item that goes in first is the item that comes out first. In programming terms, putting items in the queue is called enqueue, and removing items from the queue is called dequeue.

There are some basic operations that allow us to perform different actions on a queue:

* Enqueue: Add an element to the end of the queue.
* Dequeue: Remove an element from the front of the queue.
* Is-Empty: Check if the queue is empty.
* Peek: Get the value of the front of the queue without removing it.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

(Programiz, 2023)

***Priority Queue***

A priority queue is a type of queue that arranges elements based on their priority values. Elements with higher priority values are typically retrieved before elements with lower priority values. The hospital emergency is a real-life example of a priority queue. In this queue of patients, the patient with the most critical situation is the first in a queue, and the patient who doesn't need immediate medical attention will be the last.

In a priority queue, each element has a priority value associated with it. When you add an element to the queue, it is inserted in a position based on its priority value. For example, if you add an element with a high priority value to a priority queue, it may be inserted near the front of the queue, while an element with a low priority value may be inserted near the back.

So, a priority Queue is an extension of the queue with the following properties.

* Every item has a priority associated with it.
* An element with high priority is dequeued before an element with low priority.
* If two elements have the same priority, they are served according to their order in the queue.

A typical priority queue supports the following operations:

* Insert: When a new element is inserted in a priority queue, it moves to the empty slot from top to bottom and left to right. However, if the element is not in the correct place, then it will be compared with the parent node and the elements are swapped. The swapping process continues until all the elements are placed in the correct position.
* Deletion: It’s removed the root node from the queue, means, the one with maximum priority. This removal creates an empty slot, which will be further filled with new insertion. Then, it compares the newly inserted element with all the elements inside the queue to maintain the order.
* Peek: This operation helps to return the maximum element from queue or the minimum element from queue without deleting the node from the priority queue.

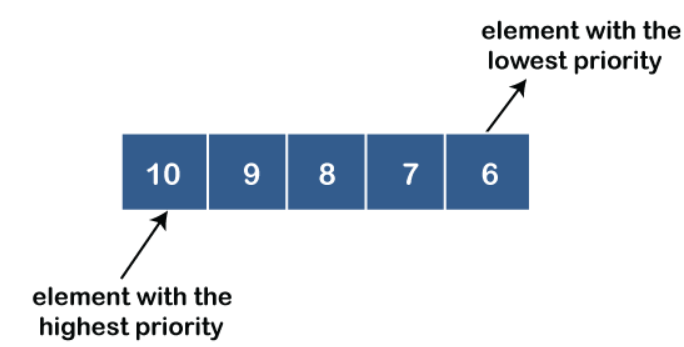
Types of Priority Queue:

* Ascending Order: As the name suggests, in ascending order priority queue, the element with a lower priority value is given a higher priority in the priority list. For example, if we have the following elements in a priority queue arranged in ascending order like 2, 6, 7, 10, 11. Here, 2 is the smallest number, therefore, it will get the highest priority in a priority queue and so when we dequeue from this priority queue, 2 will remove from the queue and dequeue returns 2.

Escala de tiempo

Descripción generada automáticamente

* Descending order: The root node is the element with the maximum priority. It will also remove the element with the highest priority first. As a result, the root node is removed from the queue.



(Geeks for Geeks, 2023)

**PHASE 3: FINDING CREATIVE SOLUTIONS**

What structures can we use to solve the problem?

From the beginning there was an idea of the data structures to be implemented, so the solution revolved around these: stack, priority queue and hash tables. In phase two, each of them was defined, making it even easier to understand its application in relation to the problem. Even so, in the following phases there will be a detailed analysis of why these specific structures were chosen over others such as arrays or lists.

**Step 4. Transition from Ideas to Preliminary Designs:**

The hash table will aim to store all the information that is provided about the task and reminders. The Priority Queue and Queue are responsible for manipulating this information with respect to the priority given to the tasks when they are assigned. And finally, the stack will be in charge of managing actions carried out in order to provide an undo option.

**Step 5. Evaluation and Selection of the Best Solution:**

Within our candidates we have three structures that are characterized by their function of storing objects. First, we have the Arrays, then a structure based on a dictionary functionality, HashTable and finally we have the LinkedList.

Criterion A. Time complexity: Element insertion time.

* [2]O(1)
* [1]O(n)

Criterion B. Temporal search complexity: Element search time.

* [2]O(1)
* [1]O(n)

Criterion C. Temporal complexity of removal: Element removal time.

* [2]O(1)
* [1]O(n)

DE criterion. LookUp Time Complexity: LookUp time of elements.

* [2]O(1)
* [1]O(n)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Criterion A | Criterion B | Criterion C | Criterion D | Total |
| Arrays | 1 | 1 | 1 | 2 | 5 |
| HashTable | 2 | 2 | 2 | 2 | 8 |
| LinkedList | 2 | 1 | 1 | 2 | 6 |

**Step 6. Preparation of Reports and Specifications:**