

Método de la ingeniería

Realizado por: Samuel Alejandro Domínguez Burbano



1 de octubre de 2023

Ingenieria de sistemas

Computación y estructuras discretas

Tabla de contenido

[Problem Identification 2](#_Toc147882127)

[Problem Definition: 2](#_Toc147882128)

[Functionalities to consider: 2](#_Toc147882129)

[Information Gathering 2](#_Toc147882130)

[Definitions: 2](#_Toc147882131)

[Search for Creative Solutions 2](#_Toc147882132)

[**Solution 1:** Coding in Java with Hash Tables and Stacks. 2](#_Toc147882133)

[**Solution 2:** Use simpler data structures, such as lists and dictionaries. 3](#_Toc147882134)

[Transition from Ideas to Preliminary Designs 3](#_Toc147882135)

[Alternative 1 Coding in Java with Hash Tables and Stacks. 3](#_Toc147882136)

[Alternative 2. Use simpler data structures, such as lists and dictionaries. 4](#_Toc147882137)

[Evaluation and Selection of the Best Solution 4](#_Toc147882138)

[Criteria 4](#_Toc147882139)

[Evaluation 5](#_Toc147882140)

[Alternative 1 Coding in Java with Hash Tables and Stacks. 5](#_Toc147882141)

[Alternative 2. Use simpler data structures, such as lists and dictionaries. 5](#_Toc147882142)

[Selection 5](#_Toc147882143)

[Preparation of Reports and Specifications 5](#_Toc147882144)

[Problem: 5](#_Toc147882145)

[Inputs: 5](#_Toc147882146)

[Output: 5](#_Toc147882147)

[Considerations: 6](#_Toc147882148)

[Flow Diagram 6](#_Toc147882149)

[Algorithm Pseudocode 6](#_Toc147882150)

[Design Implementation 6](#_Toc147882151)

[Requirement Analysis: 6](#_Toc147882152)

# Problem Identification

### Problem Definition:

A task and reminder management system should be designed to allow users to add, organize and manage their tasks and reminders.

### Functionalities to consider:

- Store tasks and reminders with the help of a hash table.

- Design an interface that allows users to add, modify and delete tasks and reminders (tasks and reminders will be sorted by due date or priority).

- Manage task priorities between priority and non-priority.

- Implement a method to undo actions performed by a user in the system.

# Information Gathering

### Definitions:

Hash table:

Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.

Stack:

A Stack is a Last in First Out (LIFO) data structure. It supports two basic operations called push and pop. The push operation adds an element at the top of the stack, and the pop operation removes an element from the top of the stack.

# Search for Creative Solutions

### **Solution 1:** Coding in Java with Hash Tables and Stacks.

A simple solution is to develop a console application in Java that allows users to manage their tasks and reminders. We can implement it like this:

We need hash table to store the tasks and reminders, where the key would be a unique identifier, the value would be an object containing information such as title, description, deadline, and priority.

We are going to design a console user interface that presents a menu with options such as "Add task or reminder", "Modify task or reminder ", "Delete task or reminder ", "View task and reminders list” and "Undo Actions".

In our application we are going to create 2 categories: "Priority" and "Non-priority". Priority tasks will be handled using a priority queue, where tasks will be automatically sorted according to their importance. Non-priority tasks will be managed in a simple list, following the FIFO (first in, first out) principle.

It implements an action logging system using a Java stack (LIFO). Each time the user performs an action, it logs the action to the stack, along with the details of the affected task and any relevant information.

The "Undo" method allows the user to revert the last action performed. This method will unstack the last action in the stack and revert the corresponding action based on the information stored in the stack.

### **Solution 2:** Use simpler data structures, such as lists and dictionaries.

Use dictionary lists to store tasks and reminders. Each dictionary represents a task or reminder and will contain information such as name, description, due date, and priority.

A user interface will be developed to allow users to add, modify and delete tasks and reminders. Users can view a list of all tasks and reminders in the list and sort them by due date or priority.

Each task can be assigned a priority field in the dictionary. To then prioritize your tasks, you can sort your task list using a custom sorting based on priority fields.

The tasks in the list are sorted according to their priority. Priority tasks are displayed first, followed by non-priority tasks. This will allow important tasks to be completed before non-priority tasks. Use lists to maintain a history of consumer activity. Each item in the manifest is a dictionary containing information about the action performed and the tasks affected. When users want to undo an action, they can use the information stored in the history to undo that action.

# Transition from Ideas to Preliminary Designs

### Alternative 1 Coding in Java with Hash Tables and Stacks.

* Easy to implement: This solution is relatively easy to implement compared to solutions that require more complex graphical interfaces. It uses a simple console interface that is easy for users to understand.
* Compliance with requirements: Meets all requirements specified in the problem description. Stores tasks and reminders in hash tables, prioritizes tasks, allows undo operations using a log and stack operations (LIFO) system, and provides basic task management functions.
* Resource efficiency: Being a console application, it uses very few system resources and is suitable for resource-constrained devices. It does not require intensive graphical interfaces or complex databases.
* Easy to use: Although it is a console application, it provides an intuitive user interface through the options menu. Users can easily perform task management operations and use the "Undo" function efficiently.
* Quick feedback: The console interface provides users with instant feedback, making it easy to track and retrieve actions performed.

### Alternative 2. Use simpler data structures, such as lists and dictionaries.

* Efficiency: As the list grows, using it to store tasks and reminders can become inefficient, especially if it needs to be searched or deleted frequently. Higher-order structures, such as hash tables and priority queues, are more efficient for certain operations.
* Code complexity: without the proposed framework, priority management and implementation of reversals would likely be more complex. This can lead to more complex and error-prone code.
* Scalability: if the system must process large amounts of data, simpler implementations may not scale and result in poor performance.

# Evaluation and Selection of the Best Solution

### Criteria

Criterion A. Accuracy of the solution. The alternative delivers a solution that satisfies everything stated in the problem:

[2] Completely

[1] Partially

Criterion B. Efficiency. A solution with better efficiency than the others considered is preferred. The efficiency can be:

[4] High

[3] Medium

[2] Low

[1] Null

Criterion C. Completeness. A solution that allows to realize all the stated requirements is preferred. How many requirements it delivers:

[3] All

[2] More than one

[1] None

Criterion D. Ease of algorithmic implementation:

[2] Makes use of simple structures

[1] Requires complex structures

### Evaluation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Criterion A | Criterion B | Criterion C | Criterion D | Total |
| Alternative 1 Coding in Java with Hash Tables and Stacks. | [2] Completely | [4] High | [3] All | [1] Requires complex structures | 10 |
| Alternative 2. Use simpler data structures, such as lists and dictionaries. | [1] Partially | [2] Low | [2] More than one | [2] Makes use of simple structures | 7 |

### Selection

According to the previous evaluation, the solution to be implemented will be alternative 1, since it obtained the highest score.

# Preparation of Reports and Specifications

Problem: Task Management and Reminder System

### Inputs:

Add New Task: the user specifies the name, description, due date, and priority of the new task.

Edit Task: The user selects an existing task and provides updated information (name, description, due date and/or priority).

Delete Task: The user selects an existing task to delete.

View Task List: User requests to view a list of all saved tasks and reminders. Sort by due date: users choose to sort tasks and reminders by due date.

Undo actions: the user chooses to undo the last action performed in the system.

### Output:

Action confirmation: after adding, changing, or deleting a task, the system will give a confirmation that the action was completed.

Task lists: when a user requests to view a list of tasks and reminders, the system displays the information for each task, including title, description, due date and priority, sorted by user request.

Undo actions: when the user chooses to undo a previous action, the system will display a message indicating the undo action and restore the corresponding task changes.

Error message: if an error occurs during the operation an appropriate error message will be displayed to inform the user.

### Considerations:

# Design Implementation

1. Store tasks and reminders in a hash table.
2. Modify a task
3. Delete a task
4. Display the task list
5. Implement a method to undo actions.

**Specifying Subroutines:**

Store Tasks and Reminders in a Hash Table

Name: addTaskReminder

Description: adds a task or reminder to the hash table.

Input:

- id (int): Unique identifier of the task.

- titulo (String): Task title.

- description (String): Task description.

- day (int): deadline day.

- month (int): deadline month.

- year (int): deadline year.

- priority (int): (if the number is 1 = "Priority", else "Not priority").

Return: True, to show that the task was done successfully.

public boolean addTaskReminder(int id, String title, String description, int day, int month, int year, int priority) {

Calendar fechaProvisional = new GregorianCalendar(day, month-1, year);

SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy");

String fechaChange = formatoFecha.format(fechaProvisional.getTime());

Priority priority1 = Priority.NO\_PRIORITY;

if (priority == 1){

priority1 = Priority.PRIORITY;

}else{

priority1 = Priority.NO\_PRIORITY;

}

Task newTask = new Task(title, description, fechaChange, priority1);

addTaskReminder(id, newTask);

if (priority == 1) {

prioritizedTasks.add(newTask);

} else {

nonPrioritizedTasks.add(newTask);

}

return true;

}

2. Modificar una tarea:

Nombre: modifyTaskReminder

Description: Modificar la data de tareas en un sistema de gestión de tareas y recordatorios.

Input:

- id (int): Unique identifier of the task.

- cambio (int): La opcion de la data a cambiar.

- modification (String): La nueva data que se le dará a la tarea.

- day (int): deadline day.

- month (int): deadline month.

- year (int): deadline year.

Return: True, para mostrar que la tarea fue modificada, False en caso contrario.

public boolean modifyTaskReminder(int id, int cambio, int day, int month, int year, String modification) {

if (tasks.containsKey(id)) {

Task task = tasks.get(id);

Task originalTask = new Task(task.getTitle(), task.getDescription(), task.getDatelimit(), task.getPriority());

switch(cambio){

case 1:

task.setTitle(modification);

actionStack.push(new Action("Modify task", originalTask, id));

return true;

case 2:

task.setDescription(modification);

actionStack.push(new Action("Modify task", originalTask, id));

return true;

case 3:

Calendar fechaProvisional = new GregorianCalendar(day, month-1, year);

SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy");

String fechaChange = formatoFecha.format(fechaProvisional.getTime());

task.setDatelimit(fechaChange);

actionStack.push(new Action("Modify task", originalTask, id));

return true;

case 4:

Priority priority1 = Priority.NO\_PRIORITY;

double priority = Integer.parseInt(modification);

if (priority == 1){

priority1 = Priority.PRIORITY;

}else{

priority1 = Priority.NO\_PRIORITY;

}

task.setPriority(priority1);

actionStack.push(new Action("Modify task", originalTask, id));

return true;

}

} else {

return false;

}

return false;

}

**3. Delete a task:**

**Name:** deleteTaskReminder.

**Description:** Deletes a task from the task list and stores it in an action stack to allow undoing the action.

**Input:**

**-** id (int): The unique identifier of the task to be deleted.

Return: None.

public void deleteTaskReminder(int id) {

        if (tasks.containsKey(id)) {

            Task task = tasks.get(id);

            Task originalTask = new Task(task.getTitle(), task.getDescription(), task.getDatelimit(), task.getPriority());

            tasks.remove(id);

            if (task.getPriority() == Priority.PRIORITY) {

                prioritizedTasks.remove(task);

            } else {

                nonPrioritizedTasks.remove(task);

            }

            actionStack.push(new Action("Delete task", originalTask, id));

        }

    }

**4. Display the task list:**

**Name:** showListTaskReminder

**Description:** Returns a string containing the list of tasks and their formatted details.

**Input:** None.

**Return:** A string representing the task list and its details.

public String showListTaskReminder() {

        String msg = "";

        for (Integer id : tasks.keySet()) {

            Task task = tasks.get(id);

            msg += "-> ID: " + id + "\n";

            msg += "╔════════════════════════════════════════════════════════════════════════════════════════════════════════\n" +

                   "║ -> Title: " + task.getTitle() + "\n" +

                   "║\n" +

                   "║ -> Description: " + task.getDescription() + "\n" +

                   "║\n" +

                   "║ -> Deadline: " + task.getDatelimit() + "\n" +

                   "║\n" +

                   "║ -> Priority: " + task.getPriority() + "\n" +

                   "╚════════════════════════════════════════════════════════════════════════════════════════════════════════\n";

        }

        return msg;

    }

**5. Implement a method to undo actions:**

**Name:** undoLastAction

**Description:** Allows to undo the last action performed by the user, if the last action was to add a task, the last added task will be deleted, if the last action was a modification, the modified task returns to its previous state and finally if the last action was to delete a task, the deleted task is recreated and entered to the hash table.

**Input:** id (int): Unique identifier of the task.

**Return:** True / False, depending on if the action was successfully undone or not.

public boolean undoLastAction() {

if (!actionStack.isEmpty()) {

Action lastAction = actionStack.pop();

String actionType = lastAction.getActionType();

Task taskDetails = lastAction.getTaskDetails();

int id = lastAction.getId();

if (actionType.equals("Add task")) {

tasks.remove(id);

if (taskDetails.getPriority() == Priority.PRIORITY) {

prioritizedTasks.remove(taskDetails);

} else {

nonPrioritizedTasks.remove(taskDetails);

}

return true;

} else if (actionType.equals("Modify task")) {

tasks.put(id, taskDetails);

return true;

} else if (actionType.equals("Delete task")) {

tasks.put(id, taskDetails);

if (taskDetails.getPriority() == Priority.PRIORITY) {

prioritizedTasks.add(taskDetails);

} else {

nonPrioritizedTasks.add(taskDetails);

}

return true;

}

}

return false;

}

# Requirement Analysis:

|  |  |
| --- | --- |
| Client |  |
| User | Students of Icesi |
| Functional Requirements | * R1: Add a task or a reminder * R2: Modify a task or a reminder * R3: Delete a task or a reminder * R4: Show the list of tasks and reminders * R5: Undo student actions |
| Problem Context | A task and reminder management system should be designed to allow users to add, organize, and manage their tasks and reminders.  It should also allow users to undo previously performed actions. |
| Non Functional requirements | * Usage of data structures such as hash, queue, and stack. * Test cases implementation to ensure the correct operating of the program. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier | R1: Add a task or a reminder | | | |
| Summary | The program must allow to add the tasks and reminders | | | |
| Input | **Input name** | **Data type** | | **Valid condition** |
| title | String | |  |
| description | String | |  |
| limitDate | Date | | dd-mm-yyyy |
| priority | Boolean | | 1 o 0 |
| id | int | | number |
| Result or Postcondition | The system verifies that the task or reminder is added successfully to the hash table and according to its priority it would then be stored in the priority queue or non-priority queue | | | |
| Output | **Output name** | | **Data type** | **Format** |
| message | | String | “The task / reminder has been successfully added” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier | R2: Modify a task or a reminder | | | |
| Summary | The system must allow the user to modify the content of an added task. | | | |
| Input | **Input name** | **Data type** | | **Valid condition** |
| selection | int | |  |
| cambio | int | | 1, 2, 3 or 4 |
| day | int | |  |
| month | int | |  |
| year | int | |  |
| modification | String | |  |
| Result or Postcondition | Depending on the data change you want to make, you enter the new data for the task | | | |
| Output | **Output name** | | **Data type** | **Format** |
| message | | String | “Successful modification” |
| message | | String | “The task could not be modified” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier | R3: Delete a task and/or reminder | | | |
| Summary | The system must allow the user to delete any of the tasks they have created. | | | |
| Input | **Input name** | **Data type** | | **Valid condition** |
| Selection | int | |  |
| Result or Postcondition | The system verifies the deletion of a task/reminder and updates the list of tasks and reminders. | | | |
| Output | **Output name** | | **Data type** | **Format** |
| N/A | | N/A | N/A |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and Name | R4: See a list of tasks and reminders | | | |
| Summary | The system must allow the user to see their list of tasks and reminders whenever they want. | | | |
| Input | **Input name** | **Data type** | | **Valid condition** |
| N/A | N/A | | N/A |
| Result or Postcondition | The system shows the list of tasks and reminders | | | |
| Output | **Output name** | | **Data type** | **Format** |
| msg | | String | List of tasks with id, title, description, due date and priority |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identifier and Name | R5: Undo actions | | | |
| Summary | The system must allow the user to undo their last action within the system. This undo functionality should revert the last action performed, whether it was adding, modifying, or deleting a task or reminder. | | | |
| Input | **Input name** | **Data type** | | **Valid condition** |
| n/a | n/a | | n/a |
| Result or Postcondition | The system reverts the last action performed by the user, if the last action was to add a task, the last added task will be deleted, if the last action was a modification, the modified task returns to its previous state and finally if the last action was to delete a task, the deleted task is recreated and entered to the hash table. | | | |
| Output | **Output name** | | **Data type** | **Format** |
| N/A | | boolean |  |