Integrative Task Engineering Method Implementation

Geoffrey Pasaje Vidal, Raúl Quigua & Juan José Angarita

ICESI

Computation & Discrete Data Structures I

Prof. Aristizabal

School of Engineering, Software Engineering

Cali, October 8th 2023

Phase 1

**Understanding Why?**

We have been given the job to create a program that let´s users deal with their daily activities and problems by allowing them to organize and sort their tasks and reminders. As of today, this app type is quite popular in the market and commonly seen in students, workers , moms and everyone else and their dog’s toolkit.This ain’t really surprising, everyday that passes seems to bring with it more and more things for us to do, so apps like this come handy when dealing with this issues. Plus productivity is trending right now. Topics like: “Maximizing your outcomes”, “optimizing your time”, “reducing distractions'', “how to effectively chase the bag” are the run-of-the-mill on social media like tik tok and instagram. Programs like Notion currently have over 30 million users worldwide, so it makes total sense why a program with these features might be needed.

**Needs**

**Defining need**

A ***"need"*** is a fundamental requirement or essential desire for something that is necessary for an individual's physical, psychological, or emotional well-being and survival. Needs can encompass a wide range of aspects, such as basic necessities like food, water, and shelter, as well as more complex needs like companionship, self-esteem, and fulfillment. Meeting one's needs is crucial for maintaining a healthy and balanced life**.**

We previously established that our program works as a task and reminder manager, that allows users to remove and add how many of these items they’d like to. It also takes into account the priority of the task ,and has special functions like a GUI and an action undo button.

**But why is it Needed?**

An individual needs to have their priorities on check. Uncertainty of what and when you gotta do something brings tons of issues like:

* **Increased Stress**: Uncertainty about what you need to do can generate stress and anxiety as you worry about forgetting important tasks or missing crucial deadlines.
* **Feeling Overwhelmed**: When you lack clarity about your tasks, it's easy to feel overwhelmed by the accumulated workload, which can lead to a sense of despair.
* **Difficulty Focusing:** Lack of structure can hinder your ability to concentrate on a specific task, leading to procrastination and reduced productivity.
* **Low Self-Esteem and Self-Efficacy:** If you can't complete your tasks due to disorganization, it can negatively impact your self-esteem and confidence in your abilities, in turn affecting your mental health.
* **Interpersonal Conflicts:** Forgetting commitments or failing to meet important personal or professional responsibilities can lead to conflicts and tensions in your relationships, negatively impacting your emotional well-being.
* **Social Isolation:** Disorganization can result in an excessive workload and a lack of time for social activities and self-care, contributing to social isolation and loneliness.
* **Impact on Sleep Quality**: Stress and worry caused by disorganization can interfere with sleep quality, which can, in turn, affect your mental health and energy levels during the day.
* **Lack of Sense of Achievement**: Not having a clear task list and failing to complete tasks can make you feel less productive and less satisfied with your accomplishments, negatively affecting your emotional well-being.
* **Mental and Emotional Exhaustion:** Disorganization can lead to mental and emotional exhaustion as you struggle to remember all your responsibilities and commitments, depleting your mental and emotional resources

Now that we are well aware of our problem we can formally define it::

**Icesi needs to develop a task management and reminder program. This program should enable users to efficiently organize and sort their daily tasks. It must provide the capability to add, remove, and prioritize tasks, along with a user-friendly graphical user interface (GUI). Additionally, an undo feature should be incorporated to correct unintended changes. The purpose of this program is to address the needs of users from various backgrounds, including students, workers, and parents, who seek effective task and reminder management.**

Specification of requirements

| ***Requirements*** | |
| --- | --- |
| ***Functional requirements*** | **RF1. Store Tasks and Reminders**  **SBRF1.1** Use a hash table to store tasks and reminders.  **SBRF1.2** Use a unique identifier as the key and task/reminder information as the value in the hash table.  **SBRF1.3** The information to store includes title, description, due date, priority, etc.  **RF2. User Interface**  Design a user interface that allows users to:  **SBRF2.1** Add new tasks and reminders.  **SBRF2.2** Modify existing tasks and reminders.  **SBRF2.3** Delete tasks and reminders.  **SBRF2.4** View a list of all tasks and reminders.  **SBRF2.5** Sort the list by due date or priority.  **RF3. Priority Management**  Priority Tasks:  **SBRF3.1** Use a priority queue to organize tasks based on their level of importance.  **SBRF3.2** Tasks are inserted into the priority queue according to their importance level when added.  Non-Priority Tasks:  **SBRF3.3** Create a category for non-priority tasks and manage them in a first-in, first-out (FIFO) order.  **RF4. Action Logging**  **SBRF4.1** Implements a stack to keep track of user actions.  **SBRF4.2** The system must store each action performed by a user.  **RF5. Undo Method**  Using a LIFO stack, the system should allow:  **SBRF5.1** Implements a method to undo the last action performed by the user.  **SBRF5.2** Provide users with an option in the user interface to use the "Undo" method and revert the last action. |
| ***Context of the problem*** | The problem context revolves around the need to develop a software system that assists users in organizing and managing their tasks and reminders effectively, addressing common challenges related to the management of pending tasks and priority administration. |
| ***Non-functional requirements*** | NRF 6. The program must have a graphic interface  NRF 7. The program must be implemented in an object oriented programming language |

Phase 2

The Research

Before we proceed with the project we must define the technicalities found in the requisites:

Definition:

*Object Oriented Programming:*

object-oriented programming (OOP) is a programming paradigm based on the concept of objects, which can contain data and code: data in the form of fields (often known as attributes or properties), and code in the form of procedures (often known as methods).

A common feature of objects is that methods are attached to them and can access and modify the object's data fields. In this brand of OOP, there is usually a special name such as this or self used to refer to the current object. In OOP, computer programs are designed by making them out of objects that interact with one another. OOP languages are diverse, but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.

Many of the most widely used programming languages (such as C++, Java,[4] Python, etc.) are multi-paradigm and they support object-oriented programming to a greater or lesser degree, typically in combination with imperative, procedural programming.

*Hash Table:*

Hash tables are a type of data structure in which the address/ index value of the data element is generated from a hash function. This enables very fast data access as the index value behaves as a key for the data value.

In other words, hash tables store key-value pairs but the key is generated through a hashing function. So the search and insertion function of a data element becomes much faster as the key values themselves become the index of the array which stores the data. During lookup, the key is hashed and the resulting hash indicates where the corresponding value is stored.

*Hash:*

A hash is a value in the table or data structure generated by the hash function used to generate that particular table or data structure. The table or data structure generated is usually called a hash table. It is also generally assumed that the time complexity of accessing data in a hash table is O(1), or constant.

*Hash Function:*

A hash function is a function that takes a set of inputs of any arbitrary size and fits them into a table or other data structure that contains fixed-size elements.

*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.

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.

*Stack:*

In computer science, a stack is an abstract data type that serves as a collection of elements, with two main operations:

* Push, which adds an element to the collection, and
* Pop, which removes the most recently added element that was not yet removed.

Additionally, a peek operation can, without modifying the stack, return the value of the last element added. Calling this structure a stack is by analogy to a set of physical items stacked one atop another, such as a stack of plates.

The order in which an element added to or removed from a stack is described as last in, first out, referred to by the acronym LIFO. As with a stack of physical objects, this structure makes it easy to take an item off the top of the stack, but accessing a datum deeper in the stack may require taking off multiple other items first.

*FIFO*:

In computing and in systems theory, first in, first out (the first in is the first out), acronymized as FIFO, is a method for organizing the manipulation of a data structure (often, specifically a data buffer) where the oldest (first) entry, or "head" of the queue, is processed first.

*LIFO:*

LIFO is an abbreviation for last in, first out. It is a method for handling data structures where the first element is processed last and the last element is processed first.

*GUI :*

The Graphical User Interface (GUI) is the way a user can interact with a computing device without entering text commands into a console. It is a friendly visual environment that allows the user to perform any action without having to have programming knowledge.

The goal of the GUI is to facilitate communication between the user and the operating system. When computer science began to develop, these first computers could only be used if you had great computer skills. But as a result of the expansion of this sector, the interface emerged. It was then that the difficulty of using operating systems or computer elements was greatly reduced. One of the pioneers in this field was Apple.

*Time and Space Complexity Analysis and Big O Notation:*

Time and space complexity analysis are methods used to measure the efficiency of algorithms. Time complexity refers to the amount of time an algorithm takes to run as a function of the length of the input. Space complexity refers to the amount of memory an algorithm requires to run as a function of the length of the input

**Big O** notation is a mathematical notation used to describe the limiting behavior of a function when the argument tends towards a particular value or infinity. It is a member of a family of notations invented by Paul Bachmann, Edmund Landau, and others, collectively called Bachmann–Landau notation or asymptotic notation. Big O notation describes the complexity of an algorithm using algebraic terms

Some key points related to Big O notation and complexity analysis are:

* Big O notation is a tool used to analyze the cost of an algorithm
* It is a mathematical analysis to provide a reference on the resources consumed by the algorithm
* Time complexity is not a measurement of how much time it takes to execute a particular algorithm because such factors as programming language, operating system, and processing power are also considered
* Asymptotic analysis is used to compare space and time complexity. It compares two algorithms based on changes in their performance as the input size is increased or decreased
* Asymptotic notations are classified into three types: Big-Oh (O) notation, Big Omega (Ω) notation, and Big Theta (Θ) notation
* Big-Oh (O) notation is used to describe the upper bound of an algorithm's time or space complexity
* Worst-case time complexity is the condition that allows an algorithm to complete statement execution in the shortest amount of time possible. In this case, the execution time serves as an upper bound on the algorithm's time complexity
* The general stepwise procedure for Big-O runtime analysis is as follows: figure out what the input is and what n represents, express the maximum number of operations the algorithm performs in terms of n, eliminate all excluding the highest order terms, and remove all the constant factors

In summary, time and space complexity analysis and Big O notation are important tools for computer scientists and software engineers to analyze the cost of an algorithm. They help to measure the efficiency of an algorithm and compare different algorithms based on their performance.

*How to Calculate Time and Space Complexity in Code:*

Calculating the time and space complexity of code is an important skill for software engineers and computer scientists. Here is a step-by-step tutorial on how to calculate the time and space complexity of code:

* **Step 1: Identify the Input Size**

The first step in calculating the time and space complexity of code is to identify the input size. The input size is the size of the data that the algorithm is processing. For example, if the algorithm is sorting an array, the input size would be the length of the array.

* **Step 2: Count the Number of Operations**

The next step is to count the number of operations that the algorithm performs as a function of the input size. An operation can be any basic step that the algorithm performs, such as a comparison or an assignment.

* **Step 3: Express the Number of Operations as a Function of the Input Size**

Once you have counted the number of operations, you need to express it as a function of the input size. For example, if the algorithm performs n comparisons to sort an array of length n, the number of operations can be expressed as O(n).

* **Step 4: Simplify the Function**

The next step is to simplify the function by removing any constant factors and lower-order terms. For example, if the function is O(2n + 3), it can be simplified to O(n).

* **Step 5: Determine the Time or Space Complexity**

Once you have simplified the function, you can determine the time or space complexity of the algorithm. The time complexity is the amount of time that the algorithm takes to run as a function of the input size, while the space complexity is the amount of memory that the algorithm requires to run as a function of the input size.

**Phase 3**

**Coming up with a Solution**

Since our client already stated which data structures we must use for the task program, our job right now is to rectify if their proposal is actually viable and a good idea. Also, it’s important to decide the framework that we’ll use to create the GUI (if we find it plausible) before we continue with the project.

***To GUI or not to GUI***

**Pros of Not Implementing a GUI in Your Java Program:**

**1. Simplicity:** If the program doesn't have a GUI, it can be simpler to design, develop, and maintain. Command-line applications are more straightforward.

**2. Resource Efficiency:** GUIs can consume more system resources (memory and CPU) than command-line programs. Not having a GUI can be beneficial if we’re working in resource-constrained environments.

**3. Faster Development:** Developing a command-line application is usually faster than creating a GUI, as it requires less design and coding for user interfaces.

**4. Automation:** Command-line programs are well-suited for automation and scripting, which can be important for certain types of tasks and for integration with other software.

**Cons of Not Implementing a GUI in Your Java Program:**

**1. Limited User-Friendliness:** Without a GUI, your program may be less user-friendly, which can make it less accessible to non-technical users.

**2. Reduced Interactivity:** Command-line programs are less interactive compared to GUIs, which can limit the user's ability to provide input or receive feedback.

**3. Complex Commands:** Users may need to memorize or look up command-line arguments and options, which can be less intuitive than clicking buttons and using graphical menus.

**4. Less Visual Appeal:** GUIs are generally more visually appealing and can provide a better user experience in terms of aesthetics.

**5. Harder to Reach a Wider Audience:** Some users prefer GUIs, and not having one may limit your program's appeal to a broader audience.

**Pros of Implementing a GUI:**

**User-Friendly:** GUIs provide a more user-friendly and intuitive way for users to interact with your program. Graphical elements like buttons, menus, and icons are easier to understand than command-line text**.**

**Improved User Experience:** GUIs can offer a visually appealing and interactive experience, making your program more engaging and easier to use**.**

**Wider User Base:** GUIs can attract a broader range of users, including those who may not be comfortable with command-line interfaces, making your program more accessible**.**

**Visual Feedback:** GUIs can provide immediate visual feedback to users, such as error messages, confirmation dialogs, and progress bars, enhancing the user experience.

**Simpler Input:** Users can input information more conveniently through text fields, checkboxes, and drop-down lists, reducing the need to remember complex commands.

**Drag-and-Drop Functionality:** GUIs allow drag-and-drop functionality, which can simplify tasks like file handling and data manipulation**.**

**Customization:** GUIs often support customizable themes and appearances, allowing users to personalize the look of the application**.**

**Cons of Implementing a GUI:**

**Development Complexity:** Creating a GUI can be more complex and time-consuming than developing a command-line application due to the need for graphical design and event handling.

**Resource Intensive:** GUI-based programs generally consume more system resources (memory and CPU) than command-line programs**.**

**Maintenance:** GUIs may require more maintenance, especially when updating the graphical elements, layouts, or adding new feature**s.**

**Learning Curve:** Developers and users might need to learn the GUI library or framework used, which can be challenging for newcomers.

**Platform Compatibility:** Ensuring cross-platform compatibility can be more challenging with GUI applications, as differences in operating systems may affect the GUI's appearance and behavior.

**Accessibility:** GUIs can pose accessibility challenges for users with disabilities, requiring extra effort to ensure compliance with accessibility standards.

Now we gotta take a look into what options we got for GUI developing in Java

**GUI Development Frameworks in Java**

**Swing**: Swing is part of the Java Foundation Classes (JFC) and is a robust and widely used GUI toolkit for Java applications. It provides a rich set of components and is platform-independent.

**JavaFX**: JavaFX is a more modern alternative to Swing for creating rich, interactive GUIs. It offers features like animations, multimedia, and a scene graph, making it suitable for both desktop and web applications.

**Java AWT (Abstract Window Toolkit)**: AWT is the oldest of these GUI frameworks and is part of the Java Standard Library. It provides basic GUI components and is lightweight, but it's less feature-rich compared to Swing and JavaFX.

**Java SWT (Standard Widget Toolkit):** SWT is an open-source widget toolkit for Java that's primarily used in Eclipse IDE. It offers native look and feel on different platforms and is known for its performance.

**Java Apache Pivot**: Apache Pivot is a platform for building web and desktop applications in Java. It focuses on a simple and productive API for GUI development.

**Java Apache JavaFX:** Apache JavaFX is an open-source distribution of JavaFX. It aims to provide the latest JavaFX features and improvements to the community.

**Java Guava:** While not a traditional GUI framework, Guava (part of the Google Guava library) provides some useful utilities for working with collections and functional programming, which can be beneficial in GUI development.

**Vaadin:** Vaadin is a Java framework for building modern web applications. It simplifies web development by allowing you to create web applications in Java, and it includes a comprehensive set of UI components.

**GWT** (Google Web Toolkit): GWT is a development toolkit for building and optimizing complex browser-based applications. It allows you to write client-side web applications in Java and then compile them into highly optimized JavaScript.

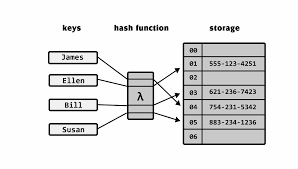
**LibGDX**: While mainly known for game development, LibGDX is a cross-platform framework that can be used for creating applications with a graphical interface. It's particularly suitable for games and multimedia applications.

***Client Idea #1 (HashTable):***

As we stated previously our client asked us to store Implement specific data structures and As Team we firmly believe the proposed ones are great ideas and we’ll emphasize on this in the following section

*Hashtable good but why?*

Implementing a HashTable in the context of the task management and reminder program is indeed a highly beneficial and practical solution. A HashTable offers a quick and efficient way to store, retrieve, and manage task-related data. Since the program needs to handle tasks and reminders with varying priorities, the HashTable can be used to associate these tasks with their respective priority levels. This allows for rapid access to tasks, ensuring that high-priority items are readily available and can be efficiently managed. Additionally, HashTables excel in minimizing search times, which is crucial when users need to access specific tasks or reminders promptly. By implementing a HashTable, the program can provide a seamless and responsive user experience, which is particularly valuable for a task management application where efficiency and user-friendliness are paramount. Moreover, HashTables can adapt to changes in the number of tasks, making it a flexible and dynamic choice, aligning perfectly with the program's requirements.



*Open Addressing (Linear Probing):*

This model allows for efficient space utilization, ensuring that we make the most of available memory. It also exhibits cache-friendly behavior, which can enhance performance, a critical factor in a task management application. Moreover, the simplicity and reliability of this approach make it an attractive choice, aligning with our goal of creating an easy-to-use and robust solution.

However, it's essential to consider potential drawbacks. Open Addressing with Linear Probing may face challenges when dealing with a high volume of tasks, potentially leading to performance issues. Additionally, collisions in the hash table could become more complex to handle compared to other collision resolution techniques. These aspects warrant careful evaluation, as we strive to strike a balance between efficiency and scalability in our final solution.

*Open Addressing (Quadratic Probing):*

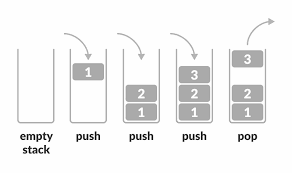
Quadratic Probing, as a collision resolution technique for our task management and reminder program, offers advantages such as efficient memory utilization and a straightforward implementation. It can be space-efficient and has cache-friendly characteristics, which align with our performance goals. However, potential issues include the risk of secondary clustering, especially under a high load factor, and the limited adaptability to dynamic changes in the dataset. These factors should be carefully considered to ensure that Quadratic Probing remains effective for our specific program requirements.

*Separate Chaining:*

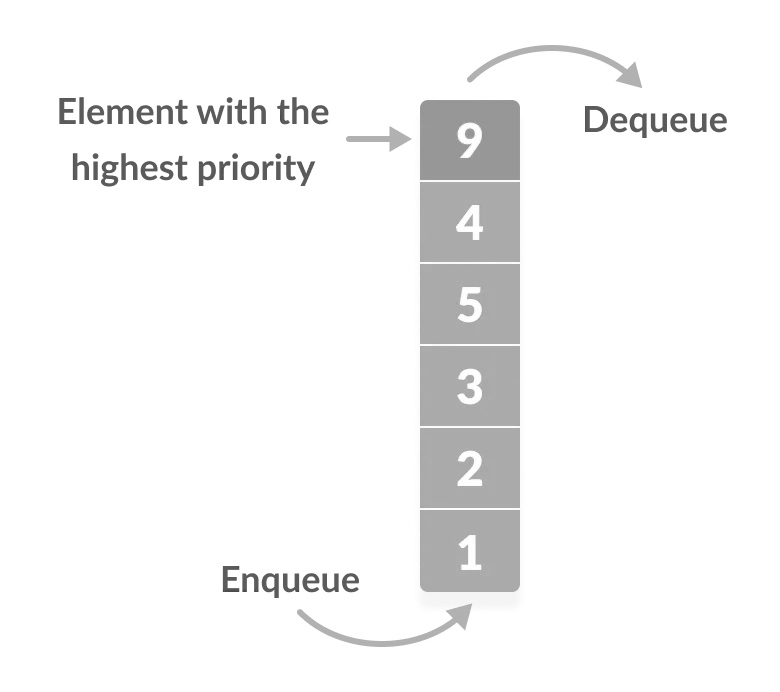
Separate Chaining stands out as an exceptionally robust collision resolution method for our task management and reminder program. Its impressive ability to efficiently handle a substantial volume of tasks ensures the program's performance remains smooth even under heavy usage. This approach shines in maintaining tasks within linked lists, offering a well-organized structure that simplifies task accessibility and management. The reliability and scalability of Separate Chaining make it an appealing choice for ensuring our program performs seamlessly and adapts to users' evolving needs.

***Client Idea #2 (Stack):***

The use of a Stack data structure proves to be highly advantageous, primarily due to the program's requirement for an "undo" action. Stacks follow the Last-In, First-Out (LIFO) principle, making them a perfect fit for tracking and managing the sequence of user actions. Whenever a user modifies or deletes a task, it's crucial to provide a straightforward way to reverse or "undo" that action. By maintaining a Stack, each user action can be pushed onto the stack, allowing for easy retrieval and reversal of the most recent action. This ensures that users can correct or backtrack their changes with a simple "undo" command, enhancing the program's usability and reducing the risk of unintended data loss or errors. Stack data structures streamline the implementation of this essential feature, making them an excellent choice for this specific context.



***Client Idea #3 (Priority Queue and Max Priority Queue):***

Using a Priority Queue and a Max Priority Queue data structure is highly beneficial. Priority Queue allows tasks to be organized based on their priority levels, ensuring that higher-priority tasks are addressed first. This is crucial for handling important tasks promptly. A Max Priority Queue specifically ensures that the most critical task is readily accessible, simplifying task management. By combining these data structures, the program can efficiently distinguish between priority and non-priority tasks, optimizing productivity and enhancing the user's ability to focus on what matters most. These structures provide an elegant solution for managing tasks with varying levels of importance in the program.

As we delve into the task program's development, we're exploring innovative ways to optimize its functionality and efficiency using different data structures. Here are some creative solutions tailored to our task management and reminder program:

1. **Stack for "Undo" Functionality**: Implementing a stack data structure proves invaluable in our task program. Stacks, following the Last-In, First-Out (LIFO) principle, are ideal for tracking and managing user actions. Users often need to undo or reverse their task-related actions, and a stack simplifies this process. Each user action can be pushed onto the stack, allowing for easy retrieval and reversal, enhancing the program's usability.
2. **Queue for Task Prioritization**: Utilizing a queue data structure can significantly enhance the task management aspect of the program. Queues, working on a first-in, first-out (FIFO) basis, can help organize and prioritize tasks effectively. Users can add tasks to the queue, and the program ensures that high-priority tasks are addressed promptly, streamlining task management.
3. **Priority Queues for Critical Tasks**: To manage tasks with varying levels of importance, the introduction of priority queues is a powerful solution. A priority queue assigns priorities to tasks, ensuring that the most critical tasks are processed first. This feature enhances the user's ability to focus on high-priority items, optimizing productivity and task management.
4. **Combining Queue and Priority Queue with Hash Table**: To create a comprehensive solution, we employ both queue and priority queue data structures in tandem with a hash table. The base information of tasks and reminders is stored in the hash table, enabling quick retrieval. The queue handles the orderly and efficient processing of standard tasks, while the priority queue ensures that urgent or high-priority tasks take precedence. This combination ensures efficient task management and seamless user interaction, aligning perfectly with the program's objectives.
5. **Java Swing Framework for User Interface:** For our task management program, we have selected Java Swing as our design framework for the user interface. Java Swing is a mature and versatile framework that provides a set of GUI (Graphical User Interface) components for building desktop applications. Its key advantages include platform independence, a wide range of customizable components, and a robust event-handling mechanism. to this include that our engineers are the most comfortable with this framework since javafx can be a little tricky

In the context of our task program, these solutions harness the strengths of various data structures to tackle specific challenges, providing an efficient and user-friendly approach to task management and reminder functionality.

Phase 4

Ideas to Preliminary Designs

In our task management program, we're presented with a range of data structures to enhance functionality. Just as in our previous example with queue and priority queue usage, our initial consideration involves the utilization of these data structures. Implementing a stack data structure for "Undo" functionality is a logical choice. Stacks, adhering to the Last-In, First-Out (LIFO) principle, are perfect for managing user actions, allowing for easy reversal of tasks. This addition significantly enhances the program's usability. In addition to this, employing a queue for task prioritization simplifies organization, following the First-In, First-Out (FIFO) principle. High-priority tasks are promptly addressed, streamlining task management. Furthermore, introducing priority queues to manage tasks of varying importance ensures the most critical tasks are processed first, optimizing productivity. The harmonious combination of these data structures with a hash table further ensures efficient task management, aligning perfectly with the program's objectives. Choosing the Java Swing framework for the user interface complements our engineers' comfort and expertise, ensuring a robust and user-friendly application.

*On the GUI design side of things…*

Implementing a GUI in your Java task program offers several compelling advantages. GUIs are inherently user-friendly, providing an intuitive way for users to interact with your application. They enhance the user experience by offering visual appeal and immediate feedback, making the program engaging and accessible to a broader audience. GUIs simplify input, allowing users to interact with the software without needing to memorize complex commands. They also support features like drag-and-drop functionality and customization, providing flexibility and personalization. While GUI development may introduce complexity, resource consumption, and maintenance efforts, the benefits in terms of user-friendliness, engagement, and accessibility make implementing a GUI a favorable choice for our program. Plus, it looks pretty cool.

It’s not a surprise for anyone that JavaSwing looks outdated by a mile. We are aware that this could be a problem since an odd looking GUI could detract potential users from trying out our program. However, with the ongoing and recent boom of the y2k cyber aesthetic and all things 2000’s I think this might be a selling point for our app.

Phase 5

Selecting the Best Solution

System Evaluation Criteria for our Task Program:

**Criterion 1**: Task and Reminder Storage

**[3]** Effectively store and manage tasks and reminders in a simple data structure.

**[2]** Basic storage functionality but may lack some features.

**[1]** Limited or no task and reminder storage.

**Criterion 2: User Interface**

**[3]** Implement a straightforward text-based user interface for user interaction.

**[2]** Basic text-based interface with room for improvement.

**[1]** No user interface, purely command-line interaction.

**Criterion 3: Task Prioritization**

**[3]** Provide a basic system for prioritizing tasks.

**[2]** Some prioritization features, but not highly customizable.

**[1]** No task prioritization system.

**Criterion 4: Task Modification and Deletion**

**[3]** Allow users to efficiently modify and delete tasks and reminders.

**[2]** Basic modification and deletion functionality but not highly efficient.

**[1]** Inefficient or no task modification and deletion.

**Criterion 5: Task Sorting**

**[3]** Implement simple sorting options for tasks based on criteria such as due date or priority.

**[2]** Limited sorting options available.

**[1]** No task sorting functionality.

**Criterion 6: Undo Functionality**

**[3]** Provide a basic "undo" feature to revert the most recent user action.

**[2]** Basic "undo" functionality but may not cover all user actions.

**[1]** Lack of "undo" feature.

**Criterion 7: Performance and Resource Efficiency**

**[3]** Ensure the program operates efficiently without significant resource consumption.

**[2]** Some performance issues or resource inefficiency.

**[1]** Poor performance and resource management.

**Now Let’s evaluate two alternatives:**

**Alternative 1:** Use Queue

**Criterion 1:** Task and Reminder Storage **[2]**

**Criterion 2**: User Interface **[3]**

**Criterion 3:** Task Prioritization **[3]**

**Criterion 4:** Task Modification and Deletion **[1]**

**Criterion 5:** Task Sorting **[1]**

**Criterion 6:** Undo Functionality **[1]**

**Criterion 7:** Performance and Resource Efficiency **[1]**

**Total score for Alternative 1:** 12

**Alternative 2:** Use Queue, Priority Queues, and Hash Table

**Criterion 1:** Task and Reminder Storage **[3]**

**Criterion 2:** User Interface **[3]**

**Criterion 3:** Task Prioritization **[3]**

**Criterion 4:** Task Modification and Deletion **[2]**

**Criterion 5:** Task Sorting **[3]**

**Criterion 6:** Undo Functionality **[2]**

**Criterion 7:** Performance and Resource Efficiency **[2]**

**Total score for Alternative 2:** 20

Based on the evaluation, Alternative 2 (Use Queue, Priority Queues, and Hash Table) clearly outperforms Alternative 1 (Use Queue) in all criteria. It scores higher in terms of task and reminder storage, task prioritization, task modification and deletion, task sorting, undo functionality, and performance/resource efficiency.

Therefore, **we’ll select Alternative 2 for our task program** as it provides a more comprehensive and efficient solution that aligns with the defined criteria.

Phase 6

Preparation of Reports and Specifications

**Problem specification:**

**Problem:** User task and reminder manager system.

**Inputs:**

Task information:

-Task Name

-Task Description

-Task Deadline

-Task Priority Level

-Task/Reminder Specification.

**Departures:**

- List of Tasks with the order of priority.

-List of Tasks with no priority

**Design:**

-You can Found the class diagram in this folder, as UMLTask.pdf

Phase 7

Implementation

Our code is here: <https://github.com/RamonaFlores/discrete-data-structures-TI1-HelloKittyInc/tree/main>

Check it out is pretty cool!

*Bibliography*

* [*Time and Space complexity in Data Structure - Ultimate Guide (simplilearn.com)*](https://www.simplilearn.com/tutorials/data-structure-tutorial/time-and-space-complexity)
* [*Time Complexity and Space Complexity - GeeksforGeeks*](https://www.geeksforgeeks.org/time-complexity-and-space-complexity/)
* [*What is Big O Notation Explained: Space and Time Complexity (freecodecamp.org)*](https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/)
* [*The Complete Guide to Big O Notation & Complexity Analysis for Algorithms: Part 1 of 2 - YouTube*](https://www.youtube.com/watch?v=HfIH3czXc-8&t=3s&ab_channel=Coderbyte)
* [*Analysis of Algorithms | Big-O analysis - GeeksforGeeks*](https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/)
* [*https://domino.ai/data-science-dictionary/hash-table*](https://domino.ai/data-science-dictionary/hash-table)
* [*https://www.umsl.edu/~siegelj/information\_theory/projects/HashingFunctionsInCryptography.html*](https://www.umsl.edu/~siegelj/information_theory/projects/HashingFunctionsInCryptography.html)
* [*https://www.arimetrics.com/en/digital-glossary/graphical-user-interface-gui*](https://www.arimetrics.com/en/digital-glossary/graphical-user-interface-gui)
* [*https://en.wikipedia.org/wiki/Stack\_(abstract\_data\_type)*](https://en.wikipedia.org/wiki/Stack_(abstract_data_type))
* [*https://en.wikipedia.org/wiki/FIFO\_(computing\_and\_electronics)*](https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics))
* [*https://www.geeksforgeeks.org/fifo-vs-lifo-approach-in-programming/*](https://www.geeksforgeeks.org/fifo-vs-lifo-approach-in-programming/)
* [*https://www.geeksforgeeks.org/lifo-last-in-first-out-approach-in-programming/*](https://www.geeksforgeeks.org/lifo-last-in-first-out-approach-in-programming/)
* [*https://www.geeksforgeeks.org/priority-queue-set-1-introduction/*](https://www.geeksforgeeks.org/priority-queue-set-1-introduction/)