SISTEMA DE GESTIÓN DE TAREAS Y RECORDATORIOS

COMPUTACIÓN Y ESTRUCTURAS DISCRETAS

JUANJO CONTRERAS

DAMY VILLEGAS

5/10/2023

METODO DEL DISEÑO DE INGENIERIA

1. IDENTIFICACION DEL PROBLEMA

El problema por desarrollar es diseñar un sistema de gestión de tareas y recordatorios que permita a los usuarios agregar, administrar y organizar sus recordatorios y tareas pendientes de una manera eficiente, considerando diferentes aspectos como la interfaz de usuario, el almacenamiento de datos, capacidad de deshacer acciones realizadas y la gestión de prioridades.

Componentes clave del problema

* Interfaz de Usuario:

La creación de una interfaz de usuario lo que nos permite hacer es agregar usuarios, modificar y eliminar tareas y recordatorios de manera intuitivas, además de esto se debe proporcionar una funcionalidad de visualización y ordenamiento de la lista de tareas y recordatorios.

* Gestión de Prioridades:

Se implementa un sistema de prioridades con dos categorías: Tareas prioritarias las cuales se gestionan mediante una cola de prioridades y Tareas no prioritaria las cuales siguen un enfoque FIFO

* Almacenamiento de Tareas y Recordatorios:

Se propone el uso de una tabla hash para gestionar y almacenar las tareas y recordatorios de los usuarios, teniendo como clave un identificador único y como valor la información relevante de las tareas o recordatorios.

* Método Deshacer

Se requiere un método que permita a los usuarios deshacer las acciones realizadas en el sistema, mediante el uso de pilas LIFO para asi llevar un registro de las acciones del usuario revirtiendo la última acción cuando sea necesario

Objetivos del sistema

En si el sistema debe ser capaz de poder almacenar tareas y recordatorios, proporcionar una interfaz de usuario amigable para la gestión de estos, organizar las tareas según su prioridad y permitir a los usuarios deshacer acciones realizadas en el sistema.

Requerimientos Funcionales del Sistema

1. R1 Almacenamiento de tareas y recordatorios:

* Utilizar una tabla hash para almacenar tareas y recordatorios.
* Cada entrada en la tabla debe contener información como título, descripción, fecha límite, prioridad, etc.

1. R2 Interfaz de Usuario:

Diseñar una interfaz de usuario que permita a los usuarios:

* Agregar nuevas tareas y recordatorios.
* Modificar tareas y recordatorios existentes.
* Eliminar tareas y recordatorios.
* Ver una lista de todas las tareas y recordatorios.
* Ordenar la lista por fecha límite o prioridad.

1. R3 Gestión de Prioridades:

* Dividir las tareas en dos categorías: "Prioritaria" y "No prioritaria".

Para tareas prioritarias:

* Utilizar una cola de prioridades para organizar las tareas según su nivel de importancia.
* Insertar nuevas tareas en la cola de prioridades según su nivel de importancia.

Para tareas no prioritarias:

* Mantener una lista de tareas no prioritarias y gestionarlas en función de su orden de llegada (FIFO).

1. R4 Función Deshacer:

* Implementar un método que permita a los usuarios deshacer la última acción realizada en el sistema.
* Utilizar una pila (LIFO) para realizar un seguimiento de las acciones realizadas por el usuario.
* Registrar cada acción realizada por el usuario en la pila, incluyendo detalles de la acción y la tarea afectada.
* Implementar un método que desapile la última acción de la pila y revierta la acción correspondiente en función de la información almacenada en la pila.

1. R5 Uso del Método de Deshacer

* En la interfaz de usuario, proporcionar a los usuarios la opción de "Deshacer" la última acción realizada.
* Cuando el usuario selecciona esta opción, llamar al método de deshacer, que revertirá la última acción realizada.

1. RECOPILACION DE LA INFORMACION NECESARIA

Se realiza una recopilación de información para resolver los problemas planteados:

* Almacenamiento de Tareas y Recordatorios

Uso de tablas Hash

* Que es una tabla Hash

Una tabla hash es una estructura de datos que asocia llaves o claves con valores, la operación principal que soporta de manera eficiente es la búsqueda: permite el acceso a los elementos (teléfono y dirección, por ejemplo) almacenados a partir de una clave generada (usando el nombre o número de cuenta, por ejemplo). Funciona transformando la clave con una función hash en un hash, un número que identifica la posición (casilla ocubeta) donde la tabla hash localiza el valor deseado.

* Como implementar una tabla Hash

Las tablas hash se suelen implementar sobre vectores de una dimensión, aunque se pueden hacer implementaciones multidimensionales basadas en varias claves. Como en el caso de los arrays, las tablas hash proveen tiempo constante de búsqueda promedio O(1),1 sin importar el número de elementos en la tabla. Sin embargo, en casos particularmente malos el tiempo de búsqueda puede llegar a O(n), es decir, en función del número de elementos.

* Que se necesita para implementar una tabla Hash

1. Una estructura de acceso directo (normalmente un array).
2. Una estructura de datos con una clave
3. Una función resumen (hash) cuyo dominio sea el espacio de claves y su imagen (o rango) los números naturales.

Fuente: <https://sites.google.com/a/espe.edu.ec/programación-ii/home/tablas-hash>

* Interfaz de Usuario

Una de las buenas prácticas de diseño de interfaz de usuario, es que debe contemplarse desde todos los niveles del proyecto. Al principio del proyecto es necesario definir una estrategia y escoger unos lineamientos (guidelines) a seguir, incluyendo los visuales: colores, tipo de letra y tamaño. Adicionalmente, las buenas prácticas que se van a seguir sí o sí.

Es importante, en la interfaz de usuario (UI por su abreviatura en inglés), identificar 3 áreas:

* Forma: Lo que se ve, la organización de cada elemento de UI.
* Contenido: La información, como se muestra y se estructura el contenido, incluye texto y multimedia.
* Comportamiento: Lo que no se ve, el comportamiento del producto, como se comunica con el usuario, difícil de identificar, pero no imposible. Ejemplo: que sucede al dar clic a un botón.

Cuando estas 3 áreas están muy bien balanceadas se puede mejorar la usabilidad del producto.

Fuente: <https://www.cognitios.co/buenas-practicas-para-diseno-de-interfaz-de-usuario-ui/>

* Gestión de Prioridades

Como implementar una cola de prioridades:

Los elementos de la cola se añaden y se eliminan de tal manera que el primero en entrar es el primero en salir. La adición de elementos se realiza a través de una operación llamada encolar (enqueue), mientras que la eliminación se denomina desencolar (dequeue). La operación de encolar inserta elementos por un extremo de la cola, mientras que la de desencolar los elimina por el otro.

El siguiente interfaz muestras las operaciones típicas para colas:

Cola.java



Fuente:

<https://www.ciberaula.com/cursos/java/colas_java.php#:~:text=Los%20elementos%20de%20la%20cola,se%20denomina%20desencolar%20(dequeue)>.

Por qué son útiles las estructuras de datos:

Las estructuras de datos son útiles porque nos permiten tener una batería de herramientas para solucionar ciertos tipos de problemas. Además, nos permiten hacer un software más eficiente optimizando recursos, algo muy útil para IoT y para los entornos que trabajan con Big Data.

Estructuras de Datos más conocidas

* Arrays

Constan de un índice para acceder a una posición concreta y del valor que el mismo almacena.

* Montículos binarios

Es una forma de guardar los datos de tal manera, que, aunque no estén ordenados, se puedan retirar de ese conjunto datos de forma ordenada.

Esto permite una gran velocidad, por ejemplo, a la hora de implementar una cola de prioridades donde queremos que cada elemento que insertemos, si insertamos de repente muchos elementos con una prioridad, el primero que se coja sea el que tenga más o menos prioridad, depende del tipo de montículo.

* Pilas

Sirven, por ejemplo, para implementar el proceso de deshacer, como cuando escribimos en un editor de texto y pulsamos CTRL+Z, lo que podemos implementar con una pila. Como vemos en la imagen, tenemos una pila vacía, el bloque 1 sería equivalente a escribir algo. El bloque 2 sería el equivalente a borrar una letra, por ejemplo. Cuando utilizamos deshacer, lo que haría sería coger la última acción realizada, que tendría una función que haría ciertas operaciones con el hecho de haber borrado una letra y la volvería a poner.

* Colas

Es otra estructura de datos muy útil, que sirve, entre otras cosas, para implementar una cola o para comunicar procesos asíncronos.

Fuente: <https://openwebinars.net/blog/que-son-las-estructuras-de-datos-y-por-que-son-tan-utiles/#:~:text=Las%20estructuras%20de%20datos%20son%20%C3%BAtiles%20porque%20nos%20permiten%20tener,que%20trabajan%20con%20Big%20Data>.

* Método Deshacer

Como implementar una pila:

Una pila es una estructura que nos permite apilar elementos y recopilarlos en el orden inverso al cual los apilamos mediante operaciones de desapilar. Esto es lo que se conoce como estructuras LIFO (Last In First Out). De esta manera una pila suele tener 3 operaciones básicas:

* APILAR: Añade un elemento a la lista.
* DESAPILAR: Retira un elemento de la lista
* ¿ESTÁ VACÍA?: Comprueba si la lista está vacía.

En el lenguaje Java contamos con la clase Stack en la librería java.util. Es importante saber que la pila hereda de la pila (aka Vector). Y que por lo tanto podemos utilizar los elementos del Vector sobre la pila.

En nuestro ejemplo vamos a insertar 10 elementos, que serán números, y los recuperaremos en su orden inverso.

Lo primero será definir la pila, es decir, instanciar la clase Stack.

Stack pila = new Stack ();

Como vemos el tipo de elementos que insertaremos en la pila serán String -cadenas-.

Una vez instanciada vamos a ir creando los elementos e insertándolos en la pila. El método que inserta elementos en la pila es. push (). Este método recibirá como parámetro el elemento a insertar.

for (int x=1;x<=10;x++)

pila.push(Integer.toString(x));

Hemos creado un bucle que nos vaya creando los números y nos hemos apoyado en la clase Integer y su método .toString() para convertir los números en cadena.

Una vez que tenemos todos los elementos, procedemos al vaciado de la pila. Tendremos que interactuar sobre la pila hasta que esta esté vacía, cosa que nos dice el método .empty(). En cada una de las iteraciones extraeremos un elemento de la pila mediante el método .pop()

while (!pila.empty())

System.out.println(pila.pop());

Pocas líneas de código que nos ayudan a crear una pila en Java.

1. BUSQUEDA DE SOLUCIONES CREATIVAS
2. Almacenamiento de tareas y recordatorios:

* Implementar una función de búsqueda que permita a los usuarios encontrar tareas y recordatorios de manera rápida y eficiente.
* Utilizar una base de datos NoSQL en lugar de una tabla hash para permitir una mayor escalabilidad y flexibilidad en el almacenamiento.
* Permitir la exportación e importación de datos para que los usuarios puedan realizar copias de seguridad y migraciones de sus tareas y recordatorios.

1. Interfaz de Usuario:

* Agregar un calendario integrado para que los usuarios puedan ver sus tareas en función de sus fechas de vencimiento.
* Ofrecer la opción de personalizar la apariencia y la disposición de la interfaz para satisfacer las preferencias individuales de los usuarios.
* Diseñar una interfaz de usuario moderna y atractiva con elementos de arrastrar y soltar para facilitar la organización de tareas.

1. Gestión de Prioridades:

* Proporcionar la capacidad de establecer recordatorios automáticos para tareas prioritarias y enviar notificaciones a los usuarios.
* Implementar un sistema de etiquetas para que los usuarios puedan etiquetar y filtrar tareas de acuerdo con sus necesidades.
* Permitir que los usuarios creen sus propias categorías de prioridades personalizadas en lugar de limitarse a "Prioritaria" y "No prioritaria".

1. Método de Deshacer:

* Ofrecer la opción de restaurar tareas y recordatorios eliminados en lugar de simplemente deshacer la acción de eliminación.
* Además de deshacer la última acción, permitir a los usuarios retroceder y deshacer múltiples acciones en secuencia.
* Proporcionar un registro de acciones deshechas para que los usuarios puedan ver y recuperar acciones específicas si es necesario.

1. TRANSICION DE LA FORMULACION DE IDEAS A LOS DISEÑOS PRELIMINARES

Ejemplo de cómo se podrían desarrollar los diseños conceptuales para el sistema de gestión de tareas y recordatorios.

1. Almacenamiento de tareas y recordatorios:

* Definir los métodos para agregar, editar y eliminar tareas y recordatorios en la base de datos.
* Crear un diagrama de entidad-relación que muestre la relación entre las tablas y cómo se almacenan los datos.
* Diseñar una estructura de base de datos que incluya tablas para tareas y recordatorios con campos para título, descripción, fecha límite, prioridad, categoría, etc.

1. Interfaz de Usuario:

* Determinar la paleta de colores, tipografía y estilo visual que se utilizará en la interfaz.
* Diseñar la navegación entre las diferentes secciones de la aplicación, como la lista de tareas, la pantalla de detalles de la tarea y la configuración.
* Realizar bocetos o wireframes de la interfaz de usuario que muestren cómo se verán las pantallas principales y cómo se organizarán los elementos.

1. Gestión de Prioridades:

* Definir cómo se manejarán las tareas no prioritarias y cómo se organizarán en una cola FIFO.
* Diseñar la interfaz de usuario para permitir a los usuarios establecer la prioridad de una tarea y cambiar su categoría.
* Especificar cómo se implementará la cola de prioridades para las tareas prioritarias, incluyendo el algoritmo de inserción y extracción.

1. Método de Deshacer:

* Crear un diagrama de flujo que describa cómo se registrarán las acciones del usuario en la pila de deshacer.
* Especificar cómo se revertirán las acciones realizadas por el usuario utilizando la pila de deshacer.
* Diseñar la interfaz de usuario para incluir una opción de "Deshacer" y mostrar mensajes de confirmación cuando se deshagan acciones.

1. EVALUACION Y SELECCIÓN DE LA MEJOR SOLUCION

Definimos la mejor solución para resolver los problemas planteados en nuestro caso lo decidimos evaluar de 1 a 5, siendo uno la menor puntuación y 5 la máxima puntuación:

Criterios para la evaluación:

Para evaluar y seleccionar la mejor solución para el sistema de gestión de tareas y recordatorios, es importante definir criterios claros y objetivos. Los criterios pueden incluir eficiencia, usabilidad, escalabilidad, mantenibilidad, requisitos del usuario, seguridad, compatibilidad y tiempo de desarrollo. A continuación, se proporciona una evaluación basada en estos criterios, junto con una puntuación del 1 al 5, donde 1 es la puntuación más baja y 5 es la puntuación más alta:

1) Almacenamiento de tareas y recordatorios:

* Solución 1 (Utilizar una base de datos NoSQL):

- Eficiencia: 4

- Usabilidad: 3

- Escalabilidad: 5

- Mantenibilidad: 4

- Requisitos del Usuario: 4

- Seguridad: 4

- Compatibilidad: 3

- Tiempo de Desarrollo: 3

* Solución 2 (Uso de Tablas Hash):

- Eficiencia: 3

- Usabilidad: 4

- Escalabilidad: 3

- Mantenibilidad: 4

- Requisitos del Usuario: 3

- Seguridad: 3

- Compatibilidad: 4

- Tiempo de Desarrollo: 4

2) Interfaz de Usuario:

* Solución 1: (Imprimir un menú en pantalla)

- Eficiencia: 3

- Usabilidad: 5

- Escalabilidad: 3

- Mantenibilidad: 4

- Requisitos del Usuario: 5

- Seguridad: 3

- Compatibilidad: 4

- Tiempo de Desarrollo: 4

* Solución 2: (Utilizar una Biblioteca grafica de Escritorio)

- Eficiencia: 4

- Usabilidad: 4

- Escalabilidad: 3

- Mantenibilidad: 3

- Requisitos del Usuario: 4

- Seguridad: 3

- Compatibilidad: 4

- Tiempo de Desarrollo: 3

3) Gestión de Prioridades:

* Solución 1 (Implementar un sistema de etiquetas):

- Eficiencia: 4

- Usabilidad: 4

- Escalabilidad: 4

- Mantenibilidad: 4

- Requisitos del Usuario: 4

- Seguridad: 3

- Compatibilidad: 4

- Tiempo de Desarrollo: 4

* Solución 2 (Implementación de una Priority Queue):

- Eficiencia: 5

- Usabilidad: 4

- Escalabilidad: 5

- Mantenibilidad: 3

- Requisitos del Usuario: 4

- Seguridad: 3

- Compatibilidad: 4

- Tiempo de Desarrollo: 4

4) Método de Deshacer:

* Solución 1 (Usar un arreglo de Controllers):

- Eficiencia: 3

- Usabilidad: 2

- Escalabilidad: 3

- Mantenibilidad: 4

- Requisitos del Usuario: 3

- Seguridad: 4

- Compatibilidad: 4

- Tiempo de Desarrollo: 3

* Solución 2 (Uso de pilas y colas):

- Eficiencia: 4

- Usabilidad: 4

- Escalabilidad: 4

- Mantenibilidad: 3

- Requisitos del Usuario: 3

- Seguridad: 4

- Compatibilidad: 5

- Tiempo de Desarrollo: 4

Después de evaluar las soluciones, podemos ver que la "Solución 2" para el almacenamiento de tareas y recordatorios y la "Solución 1" para la gestión de prioridades obtienen puntuaciones más altas en general. Por lo tanto, estas soluciones parecen ser las más prometedoras.

1. REALIZA CORRECTAMENTE EL ANÁLISIS DE COMPLEJIDAD (TEMPORAL Y ESPACIAL) PARA LOS DOS ALGORITMOS NO RECURSIVOS MÁS LARGOS Y PRESENTA SU COMPLEJIDAD EN NOTACIÓN BIG O

Los algoritmos no recursivos mas largos en el Código proporcionado son ‘put’ en la clase ‘HashTable’ y ‘enqueue’ en la clase ‘Queue’.

1. Algoritmo ‘put’ en la clase ‘HashTable’:

* Complejidad Temporal (Big O):

En el peor caso, cuando se produce una colisión en un mismo índice de la tabla hash, la búsqueda y actualización del valor asociado a una clave en la lista enlazada puede tener un tiempo lineal en el peor de los casos (O(n)), donde "n" es el número de elementos en la lista enlazada.

En promedio, si la función de hash distribuye uniformemente las claves, se espera un rendimiento mucho mejor y el tiempo promedio es constante (O(1)).

En resumen, la complejidad de put puede variar de O(1) (promedio) a O(n) (peor caso).

* Complejidad Espacial (Big O):

El espacio adicional requerido por el algoritmo put es principalmente para el almacenamiento de nodos en la tabla hash y listas enlazadas. El espacio ocupado es proporcional al número de elementos almacenados en la tabla hash, que es O(n), donde "n" es el número total de elementos almacenados.

1. Algoritmo ‘enqueue’ en la clase ‘Queue’:

* Complejidad Temporal (Big O):

La operación enqueue simplemente agrega un nuevo elemento al final de la cola, lo que es una operación constante en términos de tiempo (O(1)).

* Complejidad Espacial (Big O):

El espacio adicional requerido por el algoritmo enqueue es principalmente para almacenar el nuevo nodo en la cola. Como solo se agrega un nodo adicional, la complejidad espacial es constante (O(1)).

TABLAS TAD

TAD QUEUEABLE

|  |
| --- |
| TAD Queueable |
| 〈objeto abstracto〉: Una cola abstracta. |
| { inv: La cola debe cumplir con ciertas restricciones y propiedades específicas. } |
| ▪ enqueue(T value): Agrega un elemento a la cola.  ▪ peek(): Devuelve el elemento en el frente de la cola.  ▪ dequeue(): Elimina el elemento en el frente de la cola.  ▪ isEmpty(): Comprueba si la cola está vacía. |

TAD QUEUE

|  |
| --- |
| TAD Queue |
| 〈objeto abstracto〉: Una cola abstracta. |
| { inv: La cola debe cumplir con ciertas restricciones y propiedades específicas. } |
| ▪ getFront(): Devuelve el nodo en el frente de la cola.  ▪ setFront(QueueNode<T> front): Establece el nodo en el frente de la cola.  ▪ getBack(): Devuelve el nodo en la parte posterior de la cola.  ▪ setBack(QueueNode<T> back): Establece el nodo en la parte posterior de la cola.  ▪ enqueue(T value): Agrega un elemento al final de la cola.  ▪ peek(): Devuelve el elemento en el frente de la cola.  ▪ dequeue(): Elimina el elemento en el frente de la cola.  ▪ isEmpty(): Comprueba si la cola está vacía.  ▪ printQueue(): Imprime los elementos en la cola de frente a atrás. |

TAD PRIORITYQUEUE

|  |
| --- |
| TAD PriorityQueue |
| 〈objeto abstracto〉: Una cola de prioridad abstracta. |
| { inv: La cola de prioridad debe cumplir con ciertas restricciones y propiedades específicas. } |
| ▪ enqueue(T element): Agrega un elemento a la cola de prioridad.  ▪ dequeue(): Elimina y devuelve el elemento de mayor prioridad.  ▪ isEmpty(): Comprueba si la cola de prioridad está vacía.  ▪ printQueue(): Imprime los elementos en la cola de prioridad. |

1. DISEÑA PARA CADA OPERACIÓN DE CADA UNA DE LAS ESTRUCTURAS DE DATOS UTILIZADAS AL MENOS TRES CASOS DE PRUEBA PARA VERIFICAR SU CORRECTO FUNCIONAMIENTO (UNO ESTÁNDAR, UNO QUE PRUEBE CASOS LÍMITE Y OTRO INTERESANTE) CUMPLIENDO CON EL FORMATO DONDE SE ESPECIFICA LA CLASE, EL MÉTODO A PROBAR, EL ESCENARIO, LAS ENTRADAS Y LA SALIDA ESPERADA

* Estructura de Datos: `Hash Table`
* Método a Probar: `put ()`
* Casos de Prueba:

1.Escenario Estándar:

- Entradas: `key = "Task 1"`, `value = "Complete Assignment"`

- Salida Esperada: La entrada debe agregarse correctamente en la tabla de hash.

2. Casos Límite:

- Entradas: `key = "Task 2"`, `value = "Submit Final Project"`

- Salida Esperada: La entrada debe agregarse correctamente en la tabla de hash.

3. Caso Interesante:

- Entradas: `key = "Task 1"` (una clave que ya existe), `value = "Update Resume"`

- Salida Esperada: La entrada debe actualizarse en la tabla de hash con el nuevo valor "Update Resume" en lugar de "Complete Assignment".

* Estructura de Datos: `Queue`
* Método a Probar: `enqueue () `
* Casos de Prueba:

1. Escenario Estándar:

- Entradas: `value = "Task 1"`

- Salida Esperada: El elemento "Task 1" debe agregarse correctamente a la cola.

2. Casos Límite:

- Entradas: `value = "Task 2"`

- Salida Esperada: El elemento "Task 2" debe agregarse correctamente a la cola.

3. Caso Interesante:

- Entradas: `value = "Task 3"`

- Salida Esperada: El elemento "Task 3" debe agregarse correctamente a la cola, y la cola debe tener tres elementos en total.

* Estructura de Datos: `Priority Queue`
* Método a Probar: `enqueue () `
* Casos de Prueba:

1. Escenario Estándar:

- Entradas: `element = Task(title="Task 1", priority=PriorityLevel.HIGH, priorityOrder=3)`

- Salida Esperada: El elemento "Task 1" con alta prioridad (3) debe agregarse correctamente a la cola de prioridad.

2. Casos Límite:

- Entradas: `element = Task(title="Task 2", priority=PriorityLevel.HIGH, priorityOrder=1)`

- Salida Esperada: El elemento "Task 2" con alta prioridad (1) debe agregarse correctamente a la cola de prioridad.

3. Caso Interesante:

- Entradas: `element = Task(title="Task 3", priority=PriorityLevel.LOW, priorityOrder=-1)`

- Salida Esperada: El elemento "Task 3" con baja prioridad (-1) debe agregarse correctamente a la cola de prioridad.