TAREA INTEGRADORA 1
DISCREET GUYS INC.

INTEGRANTES:
JUAN FELIPE CASTILLO GOMEZ
JESUS DAVID RODRIGUEZ BURBANO
JUAN CAMILO RAMIREZ TABARES

DOCENTE:
ANDRES ALBERTO ARISTIZABAL PINZON

COMPUTACIÓN Y ESTRUCTURAS DISCRETAS 1 - GRUPO 1
UNIVERSIDAD ICESI
CALI - VALLE

INFORME METODO DE LA INGENIERÍA

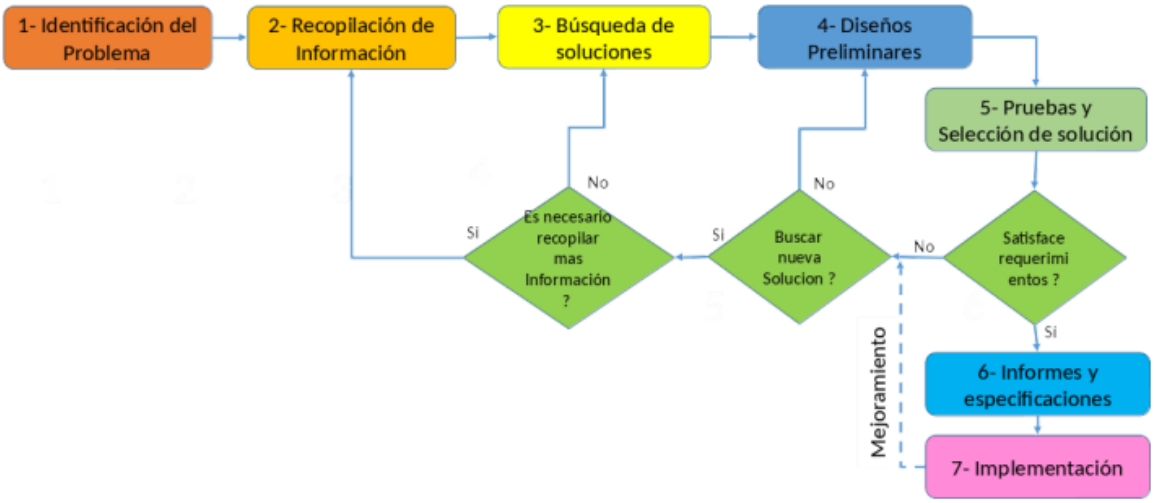
Contexto problemático

La compañía Discreet Guys Inc. de la ciudad de Cali tiene pensado construir unos nuevos edificios en el reciente lote que adquirieron cerca de la universidad ICESI, para ello requieren de una solución que les permita simular el funcionamiento de los ascensores de dichos edificios.

Desarrollo de la Solución

Para resolver la situación anterior se eligió el Método de la Ingeniería para desarrollar la solución siguiendo un enfoque sistemático y acorde con la situación problemática planteada.

Con base en la descripción del Método de la Ingeniería del libro "Introduction to Engineering" de Paul Wright, se definió el siguiente diagrama de flujo, cuyos pasos seguiremos en el desarrollo de la solución.



Paso 1. Identificación del problema

Se reconocen de manera concreta las necesidades propias de la situación problemática así como sus síntomas y condiciones bajo las cuales debe ser resuelta.

<u>Identificación de necesidades y síntomas</u>

- El ingreso de las personas al ascensor se determinará de acuerdo al orden de llegada al mismo, la salida será en orden inverso y teniendo presente las personas que se bajaran en cada piso.
- Cada ascensor se dirige a cada piso en base al orden de los botones pulsados así como también siguiendo la dirección que está tomando el ascensor.
- Cada persona podrá dirigirse a una oficina y se le brindará una de acuerdo al orden de llegada al piso.
- En caso de que la oficina a la que desea ingresar la persona esté ocupada, se le asignará un oficina libre en ese mismo piso, en el caso de que todas las oficinas del piso estén ocupadas se informará que dicha persona se quedó sin oficina.
- Cada piso contará con el mismo número de oficinas y estas estarán numeradas de forma descendente de acuerdo al piso, así pues un edificio con 3 pisos y 2 oficinas por piso contará con las oficina 6 y 5 en el piso 1, las oficinas 4 y 3 en el piso 2 y las oficinas 1 y 2 en el piso 3.

- La solución al problema deberá permitir escoger al usuario el número de edificios con los que desea trabajar, así como el número de pisos por edificio y el número de oficinas que tendrá cada piso.

Definición del Problema

La compañía Discreet Guys Inc. requiere del desarrollo de un módulo de software que permita simular el funcionamiento de los ascensores para cada edificio.

Paso 2. Recopilacion de Informacion

Con el objetivo de tener total claridad de los conceptos involucrados se hace una búsqueda de las formas de funcionamiento más utilizadas por los ascensores.

Definiciones

Fuente: https://es.wikipedia.org/wiki/Ascensor

Funcionamiento del ascensor

Para lograr un funcionamiento más eficaz, los sistemas de ascensores más modernos poseen una memoria que almacena los pedidos de llamada y los atienden priorizando las peticiones que están en dirección al coche, según distintos algoritmos de funcionamiento:

Colectiva descendente

Las botoneras colocadas en los pasillos de los pisos terminales poseen un solo botón.

En subida: El ascensor va deteniéndose en todos los pisos marcados desde la cabina, pero no atiende ninguna llamada de piso, salvo la del piso más alto por encima del último registrado por los pasajeros. Una vez llegada la cabina al último piso cuya llamada haya sido registrada, y pasado un tiempo sin nuevos pedidos, el ascensor cambia de dirección.

En bajada: El ascensor va deteniéndose en todos los pisos registrados en la cabina y también atiende los pedidos de llamada de los pisos, que supone son de bajada, hasta llegar al piso inferior que tenga un pedido de atención. En caso de que el ascensor disponga de dispositivo pesacargas el ascensor no parará en las plantas intermedias si la cabina tiene la carga completa.

Colectiva ascendente-descendente

Las botoneras colocadas en los pasillos de los pisos intermedios poseen dos botones: uno para pedidos de subida y otro para bajada.

En subida: El ascensor va deteniéndose en todos los pisos marcados desde la cabina y también en los pedidos de piso marcados como subida, pero no los de bajada. Al llegar al piso más alto por encima del último registrado por los pasajeros o desde los rellanos, y pasado un tiempo sin nuevos pedidos, el ascensor cambia de dirección.

En bajada: El ascensor va deteniéndose en todos los pisos registrados en la cabina y también atiende los pedidos de

llamada de los pisos en bajada pero no los de subida, hasta llegar al piso inferior que tenga un pedido de atención.

Paso 3. Búsqueda de Soluciones
Para este paso, se optó por buscar las estructuras de datos que mejor se acoplaban al funcionamiento del ascensor.

Alternativa 1. HashTable
Es una estructura de datos que funciona con pares de datos, un key y un value. Este key es un identificador o referencia a el value, y dicho value puede ser desde un String hasta un objeto de una clase que creemos nosotros mismos.
Las principales ventajas del hashtable son: Es una memoria de acceso aleatorio, es decir, se puede acceder a los datos que guarda directamente con el key, sin necesidad de recorrerlo por completo para hallarlo. Además de que optimiza la memoria que ocupa gracias a la función hash que implementa, el cual convierte la key en un índice para un arreglo muy pequeño.

Alternativa 2. Queues
Es una estructura de datos que es caracterizada por ser una secuencia de elementos almacenados en una lista, los cuales permiten acceder a esos datos por uno de los extremos. Los elementos se eliminan de forma en la que se van almacenando, por lo que es una estructura de tipo FIFO (first-in/first-out), esto quiere decir que el dato primero en entrar es el primero en eliminarse.

Alternativa 3. Priority Queues
Esta estructura de datos tiene un funcionamiento similar al de una Queue pero con una diferencia, el orden de salida se da por el orden de llegada y por la prioridad de cada elemento. Es decir que el primer elemento que ingresa no necesariamente es el primero que sale si no tiene una prioridad suficientemente alta.

Alternativa 4. Stacks
La estructura de datos Stacks permite apilar elementos y recopilarlos en el orden inverso, esto quiere decir que los últimos elementos en entrar son los primeros en salir. Esto es lo que se conoce como estructuras LIFO (Last In First Out).

Paso 4. Transición de las Ideas a los Diseños Preliminares
Para hacer uso de una estructura de datos que nos sirvió como ascensor tuvimos en cuenta distintos aspectos tales como: Que permitiera guardar datos y que se pudiera acceder a ellos fácilmente, Que el orden de entrada de los elementos no condicione el orden en cómo salen y por último que el orden de salida este en base a una condición que podamos especificar.

Paso 5. Evaluación y Selección de la Mejor Solución
La estructura Hashtable no fue elegida para el elevador porque la forma en la que se accede a los datos es directa y no por condiciones. Para que esta estructura de datos sirviera como elevador tendríamos que hacer muchos condicionales para simular el funcionamiento deseado, pero hay estructuras que tienen un funcionamiento base más similar al de un ascensor por lo que tenemos mejores opciones.

La estructura Queue no fue escogida porque los elementos que se van añadiendo y eliminando no necesariamente son los primeros y últimos en salir.

La estructura Priority Queue fue la elegida debido a que su funcionamiento es el más similar al de un ascensor, puesto que según el sentido en que va el elevador (hacia arriba o abajo) es como de prioridad al siguiente piso al que se dirige.

La estructura Stack no fue escogida porque los elementos que se van añadiendo y eliminando no necesariamente son los últimos en entrar y primeros en salir.

Paso 6. Preparación de Informes y Especificaciones
Para simular correctamente el funcionamiento del ascensor con esta estructura de datos necesitamos: Saber qué dato almacenaremos en nuestra priorityQueue, pues teníamos opciones como guardar personas y asignar el piso al que se dirigen como condición o prioridad, por otro lado podíamos usar los números de los pisos como datos y prioridad a la vez, esta fue la opción más viable. Además, necesitábamos poder configurar la prioridad según el sentido del elevador y que esto se haga automáticamente.

Paso 7. Implementación del Diseño
Implementación en un lenguaje de Programación:

División del problema:
1 Que el ascensor haga uso de la priorityQueue para almacenar los números de los pisos y que se defina su prioridad inicial.
2 Que el ascensor devuelva el siguiente piso al que irá según la prioridad de estos.
3 Que el ascensor pueda cambiar su prioridad según corresponda.

Construcción en un lenguaje de programación (Java):
1 En este caso la clase Elevator cuenta un atributo de tipo PriorityQueue de enteros y el piso en el que se encuentra (para facilitar otras operaciones que requieren saber el piso en el que se encuentra actualmente el ascensor). Además, el constructor inicializa la PriorityQueue como un MIN_HEAP debido a que inicialmente el ascensor se encuentra en el primer piso, por lo que recorre los piso de menor a mayor (del piso 1 al 2, luego al 3, 4, 5 …).

```java
public class Elevator{

    private PriorityQueue<Integer> elevator;
    private Integer floor;

    // por default el elevador comienta en el piso 1, por lo que crea un MIN_HEAP por default  :D
    public Elevator(){
        elevator = new PriorityQueue<Integer>(HeapTYPE.MIN_HEAP);
        floor = 1;
    }
```

2 En este caso el elevador retorna el número del piso siguiente basado en la prioridad que lleve, es decir, si está subiendo devuelve el piso más bajo que le queda para terminar su recorrido. Además de situar el piso en el que quedó.

```
private Integer leave(){
    Integer currentFloor = elevator.poll();
    floor = currentFloor;
    return currentFloor;
}
```

3 En este caso el elevador recibe la instrucción de cambiar de sentido mediante una cadena.

```
private void change(String s){
    if(s=="up"){
        elevator.setType(HeapTYPE.MIN_HEAP);
    }else if(s=="dowm"){
        elevator.setType(HeapTYPE.MAX_HEAP);
    }
}
```