**Método de la Ingeniería**

**Integrantes:**

* Cristian Perafan - A00378035
* Danna Espinosa - A00378613
* Samuel Viviescas- A00351949

**1.** **IDENTIFICACIÓN DEL PROBLEMA:**

* Discreet Guys Inc. de la ciudad de Cali necesita simular el funcionamiento de los ascensores inteligentes que implementarán en una nueva construcción de edificios cerca de la Universidad Icesi.
* La solución deberá permitir saber en qué oficina está una persona según su nombre.
* La solución al problema debe cumplir con lo siguiente:
  + El orden en que recoja a las personas se verá influenciado en el orden en el que opriman el botón y también en el sentido en que vaya el ascensor.
  + El orden de dejar a las personas en la oficina deberá ser de forma inversa a la que ingresaron al ascensor.

**2.RECOPILACIÓN DE LA INFORMACIÓN NECESARIA**

Con la intención de recordar y afianzar conocimientos procedemos a buscar información sobre las estructuras de datos que mejor se adaptan para la solución de nuestro problema.

* Pilas(Stacks):

La Stackclase representa una pila de objetos de último en entrar, primero en salir (LIFO). Extiende la clase Vector con cinco operaciones que permiten tratar un vector como una pila. Se proporcionan las operaciones habituales de empujar y sacar , así como un método para mirar el elemento superior de la pila, un método para comprobar si la pila está vacía y un método para buscar un elemento en la pila y descubrir qué tan lejos está. es de arriba.

Cuando se crea una pila por primera vez, no contiene elementos.

La interfaz y sus implementaciones proporcionan un conjunto más completo y consistente de operaciones de pila LIFO Deque, que se debe usar con preferencia a esta clase

Fuente:

<https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html>

* Colas (Queue):

Una colección diseñada para contener elementos antes del procesamiento. Además de las operaciones básicas Collection, las colas proporcionan operaciones adicionales de inserción, extracción e inspección. Cada uno de estos métodos existe en dos formas: uno lanza una excepción si la operación falla, el otro devuelve un valor especial (ya sea nulo o falso , dependiendo de la operación). La última forma de la operación de inserción está diseñada específicamente para su uso con implementaciones de Queue con capacidad restringida ; en la mayoría de las implementaciones, las operaciones de inserción no pueden fallar.

Las colas normalmente, pero no necesariamente, ordenan los elementos en una forma FIFO (primero en entrar, primero en salir). Entre las excepciones se encuentran las colas de prioridad, que ordenan los elementos según un comparador suministrado, o el orden natural de los elementos, y las colas (o pilas) LIFO que ordenan los elementos LIFO (último en entrar, primero en salir). Cualquiera que sea el orden utilizado, la cabeza de la cola es el elemento que sería eliminado por una llamada a remove() o poll(). En una cola FIFO, todos los elementos nuevos se insertan al final de la cola

Fuente:

<https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html>

* Tablas Hash (Hash Tables):

Esta clase implementa una tabla hash, que asigna claves a valores. Cualquier no null-objeto se puede utilizar como clave o como valor.

Para almacenar y recuperar con éxito objetos de una tabla hash, los objetos utilizados como claves deben implementar el hashCode método y el equals método.

Una instancia de Hashtable Tiene dos parámetros que afectan su rendimiento: la capacidad inicial y el factor de carga . La capacidad es la cantidad de cubos en la tabla hash y la capacidad inicial es simplemente la capacidad en el momento en que se crea la tabla hash. Tenga en cuenta que la tabla hash está abierta : en el caso de una "colisión de hash", un solo cubo almacena varias entradas, que deben buscarse secuencialmente. El factor de cargas es una medida de cuán llena se permite que se llene la tabla hash antes de que su capacidad aumente automáticamente. Los parámetros iniciales de capacidad y factor de carga son simplemente sugerencias para la implementación. Los detalles exactos sobre cuándo y si se invoca el método de repetición dependen de la implementación.

Fuente:

<https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html>

* ArrayList:

Implementación de matriz redimensionable de la interfaz List . Implementa todas las operaciones de lista opcionales y permite todos los elementos, incluido nulo . Además de implementar la interfaz List , esta clase proporciona métodos para manipular el tamaño de la matriz que se usa internamente para almacenar la lista. (Esta clase es más o menos equivalente a Vector , excepto que no está sincronizada).

Fuente:

<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

**3. BÚSQUEDA DE SOLUCIONES CREATIVAS**

**Posible solución #1:**

1. Usar las oficinas como tablas hash en donde las keys, sean el número de oficinas y las values serán las personas que irán a ese piso.
2. Usar una pila para recoger a las personas
3. Usar una cola de prioridad para el orden de salida del ascensor (inserción en las tablas hash)
4. Cada piso de un edificio es una cola, en donde se ingresarán a las personas según estén en el piso.

**Posible solución #2:**

1. Insertar a todas las personas de un edificio en un arrayList.
2. Usar una pila para la simulación del ascensor.
3. Usar una cola para ingresar a las personas que queden en la lista de espera
4. Insertar a las personas en la pila según su orden en el arrayList y comparando el piso en el que está, por ejemplo:*El ascensor se dirige al piso con base al orden en que los usuarios pulsen el botón, si la primera persona está en el segundo piso y oprime el botón, el ascensor subirá desde el piso 1 al piso 2 y lo recogerá, si la segunda persona está en el piso 1 y oprime el botón, el ascensor la dejará en la cola de espera, debido a que el ascensor está subiendo, e irá por la tercera persona, así hasta que recoja a todas las personas que estén en el mismo o un piso más arriba de donde está el ascensor.Cuando recoja a todas las personas, irá por las personas que están abajo, es decir bajará a recoger a todas las personas que quedaron en la cola de espera.*
5. Insertar a las personas en las oficinas (hashmap) de forma inversa a la que entraron en el ascensor, es decir, hacer pull en la pila, y agregar a la persona a la oficina a la que tiene de destino.
6. Finalmente, para la búsqueda, se pide el identificador del edificio, luego el numero de oficina y se busca en las tablas hash con el método get.

**Posible solución #3:**

1. Hacer un ordenamiento de menor a mayor de las personas según los pisos en los que están
2. Agregar a las personas a un arrayList según el ordenamiento anterior, es decir ingresará primero a las personas del primer piso hasta el último piso y después pasará cada persona a la oficina de destino en la tabla hash

**4. TRANSICIÓN DE LA FORMULACIÓN DE IDEAS A LOS DISEÑOS PRELIMINARES**

Debido a que en la posible solución #3 no se usan las estructuras de datos que nos exigen en el planteamiento del problema, descartamos esta opción.

**5. EVALUACIÓN Y SELECCIÓN DE LA MEJOR SOLUCIÓN**

**Criterio A. Precisión de la solución**

[2] Completa

­ [1] Parcial

**Criterio B. Eficiencia.**

­ [4] Constante

­ [3] Mayor a constante

­ [2] Logarítmica

[1] Lineal

**Criterio C. Completitud.**

­ [3] Todas

­ [2] Más de una si las hay, aunque no todas

­ [1] Sólo una o ninguna

**Criterio D. Facilidad en implementación algorítmica.**

[2] Compatible con las estructuras de datos de java.

[1] No compatible completamente con las estructuras de datos de java.

|  | **Criterio A** | **Criterio B** | **Criterio C** | **Criterio D** | **Total** |
| --- | --- | --- | --- | --- | --- |
| **Solución 1** | **(1)** | **(1)** | **(2)** | **(2)** | **6** |
| **Solución 2** | **(2)** | **(1)** | **(3)** | **(2)** | **8** |

De acuerdo con la evaluación anterior, la solución más fiable es la 2 ya que obtuvo mayor puntaje de acuerdo a los criterios definidos.

**6. PREPARACIÓN DE INFORMES Y ESPECIFICACIONES**

**R1.Ingreso al ascensor:**

El sistema permite el ingreso de las personas al ascensor de acuerdo al orden de llegada.

**R2. Salida del ascensor:**

El sistema permite que las personas salgan del ascensor en el orden inverso al que ingresaron.

**R3.Búsqueda de persona en el ascensor:**

El sistema debe permitir la búsqueda de una persona a partir de un nombre ingresado y retornar el número de la oficina en donde se encuentra.

**(Los diagramas se encuentran en el documento de especificación de requerimiento)**

**7.IMPLEMENTACIÓN DEL DISEÑO**

Lista de tareas a implementar:

1. Iniciar programa
2. Menú 1 ciclo.
3. Mostrar menú 1 y seleccionar opción.
4. Selección de opción
5. Preguntar entradas (Número de edificios, Nombre Edificio, Número de Pisos, Número de Oficinas, Número de personas en el edificio.)
6. Agregar edificios a un arraylist de edificios.
7. Mostrar menú 2 (Menú de simulación) y seleccionar opción
8. Selección de opción menú 2
9. Comienzo de simulación
10. Mostrar la información del edificio
11. Consultar si una persona está dentro de una oficina.

…..

**Construcción:**

A.

public Main() {

informationRegistered = false;

sc = new Scanner(System.in);

controller = new Controller();

}

B.

public static void main(String [] args ) {

Main pc = new Main();

System.out.println("Starting the APP...");

int option = 0;

do {

option = pc.showMenu();

pc.executeOperation(option);

}while(option != 0);

}

C.

public int showMenu() {

System.out.println("\*\*\* MENU \*\*\*\n"+

"(1) Enter data to simulate the operation of the building\n"+

"(2) Simulation\n"+

"(0) Exit");

int option = sc.nextInt();

sc.nextLine();

return option;

}

D.

public void executeOperation(int option) {

switch(option) {

case 0:

System.out.println("Bye, see you soon");

break;

case 1:

readSimulationInformation();

break;

case 2:

if(informationRegistered == true) {

ShowSimulationMenu();

}

else {

System.out.println("You must first register the information of the simulation!!");

}

break;

}

}

E.

public void readSimulationInformation() {

System.out.println("\*\*\* SIMULATION INFORMATION \*\*\*\n"+

"Enter simulation data");

int numberOfBuildings = sc.nextInt();

sc.nextLine();

for (int i = 0; i < numberOfBuildings; i++) {

String buildingInformation = sc.nextLine();

String [] separetedBuildingInformation = buildingInformation.split(" ");

String identifier = separetedBuildingInformation[0];

int numPeople = Integer.valueOf(separetedBuildingInformation[1]);

//Auxiliary array to save the people information at the building

Person [] peopleAtBuilding = new Person[numPeople];

int amountFloors = Integer.valueOf(separetedBuildingInformation[2]);

int numOfficesPerFloor = Integer.valueOf(separetedBuildingInformation[3]);

for(int j = 0;j<numPeople;j++) {

String personInformation = sc.nextLine();

String [] separatedPersonInformation = personInformation.split(" ");

String name = separatedPersonInformation[0];

int floorPersonIs = Integer.valueOf(separatedPersonInformation[1]);

int floorPersonGoes = Integer.valueOf(separatedPersonInformation[2]);

peopleAtBuilding[j] = new Person(name,floorPersonIs,floorPersonGoes);

}

F.

public void addABuildingToAList(Building obj) {

buildings.add(obj);

}

……

**Los demás métodos se encuentran implementados en el código.**