MÉTODO DE LA INGENIERÍA: APLICACIÓN PARA LA SOLUCIÓN DE LA TAREA INTEGRADORA 1

Rodríguez Steban Amilcar, Plaza Juan Felipe y Jojoa Jorge Eduardo.

Universidad Icesi, Facultad de Ingeniería, Departamento de TIC, Computación y estructuras discretas 1.

Santiago de Cali, 3 de abril del 2022.

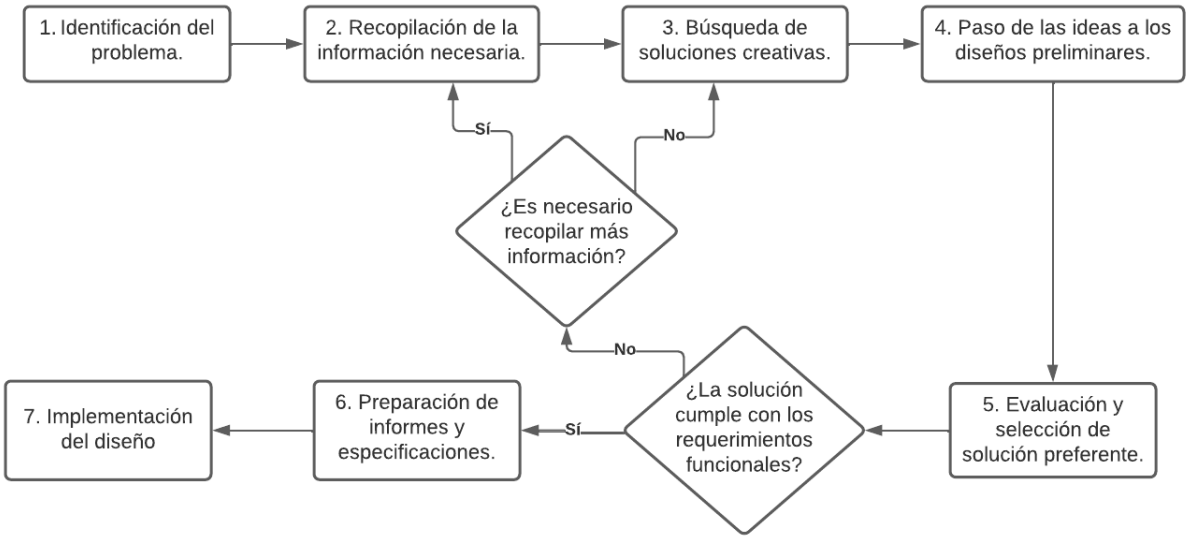
Contexto problemático:

La compañía Discreet Guys Inc[1], contrata a su equipo técnico, con el objetivo de prestar el servicio de simulación al funcionamiento de los ascensores de los nuevos edificios que van a construir en el reciente lote que adquirieron cerca de la universidad ICESI. En efecto, la compañía requiere una solución que simule el funcionamiento dentro de los nuevos 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.

Teniendo en cuenta la definición del método de la ingeniería del libro "Introducción a la Ingeniería" de Paul Wright, se definió el siguiente diagrama de flujo, cuyos pasos se seguirán para el desarrollo de la solución.



[1] Compañía ficticia.

Paso 1. Identificación del problema

Se distinguen de forma concreta las necesidades de la situación problema y las condiciones bajo las cuales debe ser resuelta.

Identificación de las necesidades de la situación problema

- Se debe garantizar la correcta simulación del movimiento del ascensor y de las personas dentro de cada edificio que la empresa Discreet Guys Inc va a construir.

- Cada edificio cuenta con un número determinado de oficinas y personas. Al final de cada simulación, cada persona va a ocupar una oficina.

- Las personas se moverán a través de los pisos por un ascensor. El ingreso de las personas al ascensor se determinará de acuerdo con el orden de llegada al ascensor. La salida será en el orden inverso.

- Cada ascensor se dirige a cada piso con base al orden en el que los usuarios pulsan el botón. El orden se ve afectado por la dirección en la que se dirige el ascensor.

- La solución al problema debe atender diferentes entradas con diversos casos de prueba.

Definición del problema

La empresa Discreet Guys Inc de la ciudad de Cali requiere simular el funcionamiento de los ascensores de los nuevos edificios que van a construir.

Paso 2. Recopilación de la Información

Para el planteamiento de la solución, es necesario conocer los conceptos involucrados en el desarrollo de esta. Por ende, a continuación, se hará énfasis en las definiciones de algunos tipos de estructuras de datos que se consideran, anticipadamente, harán parte de una posible solución final. Cabe resaltar, las definiciones que se presentan son tomadas de fuentes confiables y reconocidas.

Definiciones

Fuente:

https://www.informatica-juridica.com/wp-content/uploads/2014/01/Documentacion_Biblioteca_Estructuras_Datos_Avanzadas.pdf

Lista

Una lista se define como una n-tupla de elementos (donde ***li*** es el i-ésimo elemento de la lista) ordenados de forma consecutiva, o sea, el elemento ***li*** precede al elemento $li + 1$: $L = (l1, l2, \dots, ln)$. Si la lista contiene cero elementos se denomina lista vacía.

Tablas Hash

Es una estructura de datos que está formada por las combinaciones de llaves o claves con valores organizados en "sectores de almacenamiento", para permitir realizar una búsqueda rápida. Constituye un TDA especial para la manipulación y almacenamiento de la información en la memoria secundaria de la computadora

Pila

Una pila (stack en inglés) es una estructura sencilla, mucho más simple que la lista y se puede definir como una colección ordenada de elementos, $S = (S_1, S_2, \dots, S_n)$, donde se adicionan y eliminan por un mismo extremo conocido como tope. Se dice que S_1 es el elemento del fondo de la pila y S_n es el elemento que se encuentra en el tope.

Cola

Una cola (queue en inglés) es una estructura de datos caracterizada por ser una lista ordenada de elementos$Q = (Q_1, Q_2, \dots, Q_n)$, donde la operación de adición se realiza por un extremo, llamado cola, y la operación de extracción por el otro, llamado cabeza.

Cola de prioridad

Una cola de prioridad es una cola cuyos elementos tienen asociado una prioridad y se atienden en el orden indicado por la misma, de forma que el orden en que los elementos son procesados sigue las siguientes reglas:

- El elemento con mayor prioridad es procesado primero.
- Si varios elementos tienen la misma prioridad, se atenderán en dependencia del orden en que fueron adicionados.

Paso 3. Búsqueda de Soluciones Creativas

En primer lugar, se propone hacer un programa en el lenguaje y versión Java SE 8 con una interfaz por consola que, a través de una clase Menú, reciba los datos para realizar la simulación, como lo son: la cantidad de edificios, identificador para cada edificio, el número de personas que se encuentra de momento en el edificio, la cantidad de pisos, la cantidad de oficinas por piso, el nombre de cada persona que está en el edificio, el número de piso donde ella se encuentra y el número de oficina donde la persona se quiere dirigir.

En segundo lugar, se propone crear una clase edificio que contenga una pila de personas, una lista de oficinas y una lista de oficinas. La pila de personas se propone dado a que se infiere

Finalmente, se plantea hacer una clase ascensor que implemente los métodos de una clase tipo interfaz, tales como subir y bajar. Sin embargo, se evalúan tres ideas para la forma en que entran y salen las personas al ascensor:

- ***Alternativa A. Pila:***

Las personas que ingresen al ascensor se van a ir guardando a través de una pila. Lo anterior, dado a que una vez el ascensor haya recogido todas las personas independientemente de la dirección en que se dirija, las va a ir retirando una a una en cada piso comenzando desde la última persona que recogió.

- ***Alternativa B. Cola:***

Las personas que ingresen al ascensor se van a ir guardando a través de una cola. Lo anterior, dado a que una vez el ascensor haya recogido todas las personas independientemente de la dirección en que se dirija, las va a ir retirando una a una en cada piso comenzando desde la primera persona que recogió.

- ***Alternativa C. Cola de prioridad:***

Las personas que ingresen al ascensor se van a ir guardando a través de una cola de prioridad. Lo anterior, dado a que el ascensor debe recoger a todas las personas dependiendo la dirección en que se dirija y las va a ir retirando una a una en cada piso comenzando desde la primera persona que recogió.

Paso 4. Transición de la Ideas a los diseños preliminares

La revisión de las alternativas presentadas en el paso anterior nos conduce a lo siguiente:

- ***Alternativa A. Pila:***

    - La primera persona que entra al ascensor será la última en ser retirada en el piso que seleccionó. En otras palabras, no atiende la necesidad de la prioridad que requiere el programa.

- ***Alternativa B. Cola:***

    - La primera persona que entra al ascensor será la primera en ser retirada en el piso que seleccionó. Esta opción si atiende la prioridad de la primera persona; sin embargo, no le presta atención a la forma en que va a recoger a las personas, dado a que depende de la dirección en que se dirige el ascensor. Por lo tanto, atiende la necesidad de la primera persona, pero no atiende la necesidad de las personas que ingresen después de ella al ascensor.

- ***Alternativa C. Cola de prioridad:***

    - La primera persona que entra al ascensor será la primera en ser retirada en el piso que seleccionó. Además, dependiendo el orden de llamado del ascensor y la dirección en que se dirija, va a recoger y retirar a las personas en los pisos que seleccionaron. Por lo tanto, esta alternativa atiende a la necesidad de todas las personas que ingresen al ascensor.

Paso 5. Evaluación y selección de la mejor solución.

Para comenzar con este paso del método, cabe resaltar que ya se tiene una base para comenzar el proyecto. Por otro lado, la clase Ascensor es aquella que presenta tres alternativas para ser evaluadas.

Teniendo en cuenta las necesidades del problema, para comenzar con la implementación del diseño, en primer lugar, se descartan las alternativas que nos son viables para la establecer una solución. En este caso, se descartan las alternativas A y B, dado a que no atienden a la necesidad de recoger a las personas en cada piso considerando la dirección en que se dirija el ascensor. Por lo tanto, la revisión cuidadosa de las tres alternativas nos conduce a la siguiente para el desarrollo del diseño preliminar:

Alternativa C. Cola de prioridad.

Se decide esta opción, puesto que, si el ascensor es llamado por una persona que desea subir de piso, primero va a recoger a todas las personas que desean subir y, posteriormente, va a recoger, dependiendo el orden de llamado, a todas las personas que deseen bajar de piso.

Paso 6. Preparación de Informes y Especificaciones

Especificación del Problema (en términos de entrada y salida)

- Problema:
  Simulación del movimiento del ascensor y de las personas dentro de un edificio.

- Entradas:
  - Cantidad de Edificios.
  - Identificador de cada edificio.
  - Numero de personas en el edificio.
  - Cantidad de pisos.
  - Cantidad de oficinas por piso.
  - Nombre de cada persona que está en el edificio.
  - Número de piso donde cada persona se encuentra.
  - Número de oficina donde cada persona se quiere dirigir

- Salida:
  Movimiento de los ascensores y personas en cada edificio.

Paso 7. Implementación del Diseño

Implementación en Java SE 8.

Lista de tareas a implementar:

a) Crear un edificio.
b) Crear una persona.
c) Recoger una persona
d) Retirar una persona.
e) Poner una persona en una oficina.

Especificación de Subrutinas

Construcción

Escritura del código en un Lenguaje de Programación. Java en este caso

Crear un edificio

Nombre:	createBuilding
Descripción:	Crea un edificio con personas y oficinas.
Entrada:	- buildName: String, es el nombre del edificio. - numFloors: int, es el número de pisos del edificio. - numOffices: int, es el número de oficinas del edificio.
Retorno:	void

```
public void createBuilding(String buildName,
        int numFloors, int numOffices) {
    Edificio temp =
            new Edificio(buildName, numFloors, numOffices);
    building.add(temp);

}
```

Crear una persona

Nombre:	createPerson
Descripción:	Crea una persona dentro de un edificio.
Entrada:	- perName: String, es el nombre de la persona. - start: int, es el piso donde se encuentra la persona. - destiny: int, es el número de oficina a donde se dirige. - buildName: int, es el identificador de cada edificio.
Retorno:	void

```
public void createPerson(String perName,
        int start, int destiny, int buildName) {
    Person temp =
            new Person(perName, start, destiny);
    building.get(buildName).getPersonsList().add(temp);
}
```

Recoger una persona

Nombre:	pickUpPerson
Descripción:	Recoge una persona de un piso.
Entrada:	- buildName: int, es el identificados del edificio d.
Retorno:	void

```
public void pickUpPerson(int buildName) {
    if (transporte.getPila().size() < building.get(buildName).getPersonsList().size()) {
        Person aux = personQueue.lastElement();
        if (building.get(buildName).getPersonsList().size() == personQueue.size()
                || (building.get(buildName).getPersonsList().size() - 1) == personQueue.size()) {
            if (transporte.getCurrentFloor() < aux.getcurrentFloor()) {
                transporte.subir(aux.getcurrentFloor());
            } else {
                transporte.bajar(aux.getcurrentFloor());
            }
            personQueue.pop();
            transporte.entrarAscensor(aux);
            pickUpPerson(buildName);
        } else {
            if (transporte.isComingDown() && transporte.getCurrentFloor() > aux.getcurrentFloor()) {
                transporte.bajar(aux.getcurrentFloor());
                personQueue.pop();
                transporte.entrarAscensor(aux);
                if (transporte.getCurrentFloor() == 0) {
                    transporte.subir(0);
                }
                pickUpPerson(buildName);
            } else if (!transporte.isComingDown() && transporte.getCurrentFloor() < aux.getcurrentFloor()) {
                transporte.subir(aux.getcurrentFloor());
                personQueue.pop();
                transporte.entrarAscensor(aux);
                if (transporte.getCurrentFloor() == building.get(buildName).getOffices().size() - 1) {
                    transporte.bajar(building.get(buildName).getOffices().size() - 1);
                } pickUpPerson(buildName);
            } else {
                personQueue.pop();
                personQueue.add(0, aux);
                pickUpPerson(buildName);
            }
        }
    }
}
```

Retirar una persona

Nombre:	getOutPerson
Descripción:	Retira a una persona del ascensor.
Entrada:	- buildName: int, es el identificados del edificio d.
Retorno:	void

```
public void getOutPerson(int buildName) {
    if (transporte.getPila().size() < building.get(buildName).getPersonsList().size()) {
        Person aux = transporte.getPila().firstElement();

        if (building.get(buildName).getPersonsList().size() == transporte.getPila().size()
                || (building.get(buildName).getPersonsList().size() - 1) == transporte.getPila().size()) {
            if (transporte.getCurrentFloor() < edificio.searchFloorOffice(aux.getDestination(),
                    aux.getcurrentFloor())) {
                transporte.subir(edificio.searchFloorOffice(aux.getDestination(), aux.getcurrentFloor()));
            } else {
                transporte.bajar(edificio.searchFloorOffice(aux.getDestination(), aux.getcurrentFloor()));
            }
            transporte.salirAscensor();
            getOutPerson(buildName);
        }
    }
}
```

Retirar una persona

Nombre:	finalState
Descripción:	Pone a la persona en una oficina del edificio.
Entrada:	- buildName: int, es el identificados del edificio d.
Retorno:	void

```java
public String finalState() {
    String message="[";
    int cont=0;
    int indexOffice=1;
    for(int i=offices.size()-1;i>=0;i--) {
        for(int j=0;j<offices.get(i).size();j++) {
            if(offices.get(i).get(indexOffice)!=null) {
                cont++;
                if(indexOffice==(officesPerFloor*floors)||cont==personsList.size()) {
                    message+=offices.get(i).get(indexOffice).getName();
                }
                else {
                    message+=offices.get(i).get(indexOffice).getName()+", ";
                }
            }
            indexOffice++;
        }
    }
    message+="]";
    return message;
}
```