## Especificación del Problema

|  |  |
| --- | --- |
| **Cliente** | Discreet Guys Inc. |
| **Usuario** | * Personas |
| **Requerimientos funcionales** | 1. El programa debe de determinada a cada persona hacia dónde se dirige (esto es piso y oficina). 2. El programa debe simular el funcionamiento de los ascensores. Es importante saber que el ingreso de las personas a los ascensores se determinará de acuerdo con el orden de llegada. La salida será en el orden inverso. 3. El programa debe entregar un “.txt” con el nombre y la posición final de cada persona por edificio. |
| **Contexto del problema** | Discreet Guys Inc. de la ciudad de Cali decide contratar a su equipo para simular el funcionamiento de los ascensores de los nuevos edificios que van a construir en el reciente lote que adquirieron cerca de la universidad ICESI.  Estos edificios contarán con varias oficinas en cada piso, siendo el número de ellas igual por cada piso del edificio al que se esté referenciando en la entrada. Además, cada oficina contará con un número que la identifica en orden ascendente a medida que se desciende por el edificio, es decir, si hay dos pisos y dos oficinas por piso entonces en el piso 2 estarán las oficinas 1 y 2, y en el piso 1 estarán las oficinas 3 y 4. Por último, en cada piso de un edificio se encontrará un número determinado de personas (este puede variar por piso).  El ingreso de las personas a los ascensores se determinará de acuerdo con el orden de llegada al ascensor. La salida será en el orden inverso. |
| **Requerimientos no funcionales** | * Uso de las estructuras estudiadas en la clase. * Uso de Generics. |

## Especificación de los Requerimientos

### Requerimiento Funcional 1

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre o identificador** | R1: El programa debe de determinada a cada persona hacia dónde se dirige (esto es piso y oficina). | | |
| **Resumen** | El programa debe de leer la entrada de un documento “.txt” y separar la información por edificio y personas. De la información de las personas identificar cual es la posición final y realizar el traslado de las personas. En caso de la oficina estar llena, el programa debe de calcular y asignar una nueva oficina a la persona. Si todas las oficinas están llenas, debe de indicar y guardar eso como posición final. | | |
| **Entradas** | **Nombre entrada** | **Tipo de dato** | **Condición de selección o repetición** |
| nombre | String | Que sea obtenido (leído) del archivo “.txt” |
| Piso | Int | Que sea obtenido (leído) del archivo “.txt” |
| Oficina | Int | Que sea obtenido (leído) del archivo “.txt” |
| **Actividades generales necesarias para obtener los resultados** | 1. El programa lee el archivo “.txt” con la información de:  * Número de edificios. * Código de edificio, número de personas, número de pisos y número de oficinas por piso. * Información de cada persona (posición actual y oficina final).  1. Identifica cual es la posición oficina final para cada persona. 2. Empieza el funcionamiento de los ascensores (R2). 3. En caso de la oficina estar ocupada, indicar que la persona quedara fuera del edificio. | | |
| **Resultado o Postcondición** | Todas las personas quedan ubicadas en la posición final. | | |
| **Salidas** | **Nombre salida** | **Tipo de dato** | **Condición de selección o repetición** |
| N/A | N/A | N/A |

### Requerimiento Funcional 2

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre o identificador** | R2: El programa debe simular el funcionamiento de los ascensores. Es importante saber que el ingreso de las personas a los ascensores se determinará de acuerdo con el orden de llegada. La salida será en el orden inverso. | | |
| **Resumen** | El programa debe simular cómo funciona el ascensor. Es importante tener en cuenta que el orden de bajada del ascensor depende de la entrada. Además, tiene prioridad de si está subiendo o bajando. Por ejemplo: El ascensor está en el piso 2 y va subiendo, si el usuario A ingresa y presiona primero el 1 y luego el usuario B el 3, obviamente llevará primero al usuario B. | | |
| **Entradas** | **Nombre entrada** | **Tipo de dato** | **Condición de selección o repetición** |
| persona | Persona | El edifico esta hecho y tiene personas |
| **Actividades generales necesarias para obtener los resultados** | 1. Ingresan las personas al ascensor que es esta realizado con una cola de prioridad. 2. Llega a nuevo piso y empieza a bajar a las personas y a introducirlas en otra cola de prioridad identificando si no se bajan en ese piso. 3. vuelve a moverse a otro piso y repite el paso 2 pero en sentido opuesto. 4. Revisa que el ascensor este básico y en caso de ser así, regresa al primer piso. | | |
| **Resultado o Postcondición** | El ascensor queda vacío en el primer piso y todas las personas ubicadas en su posición final. | | |
| **Salidas** | **Nombre salida** | **Tipo de dato** | **Condición de selección o repetición** |
| N/A | N/A | N/A |

### Requerimiento Funcional 3

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre o identificador** | R3: El programa debe entregar un “.txt” con el nombre y la posición final de cada persona por edificio. | | |
| **Resumen** | El programa debe generar un archivo “.txt” para esto debe volver a leer el orden en como quedaron las oficinas de cada uno de los edificios y guardarla. Generar un archivo “.txt” con la información del estado final de los edificios. | | |
| **Entradas** | **Nombre entrada** | **Tipo de dato** | **Condición de selección o repetición** |
| Reporte | String | Que el proceso le lectura y reorganización de las personas fue exitoso. |
| **Actividades generales necesarias para obtener los resultados** | 1. Revisa que el String no este vacío. 2. Genera un archivo “.txt” con la información anterior | | |
| **Resultado o Postcondición** | Un archivo “.txt” con la información del estado final cada uno de los edificios, las oficinas y las personas. | | |
| **Salidas** | **Nombre salida** | **Tipo de dato** | **Condición de selección o repetición** |
| Output | .txt | El String “Reporte” no es nulo |

**Complejidad temporal:**

Algoritmo 1)

*public void addBuilding(String code, int floors, int officesPerFloor, int people) {*

*Building newBuilding = new Building(code, floors, officesPerFloor, people);* **1**

*for (int i = 0; i < buildings.length; i++) {* **n+1**

*if (buildings[i] == null) {* **n**

*buildings[i] = newBuilding;* **1**

*break;* **1**

*}*

*}*

*}*

|  |  |
| --- | --- |
| Linea | # |
| 1 | 1 |
| 2 | n+1 |
| 3 | n |
| 4 | 1 |
| 5 | 1 |

Algoritmo 2)

*public void queuePersonInFloor(String name, int floor, int destination) {*

*Person p = new Person(name, floor, destination);* **1**

*if (floorQueues[floor] != null) {* **1**

*floorQueues[floor].add(p);* **1**

*} else {*

*Queue<Person> q = new Queue<>();* **1**

*q.add(p);* **1**

*floorQueues[floor] = q;* 1

*}*

*}*

|  |  |
| --- | --- |
| *Linea* | *#* |
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |