# Método de Ingeniería

**a) Identificación del Problema:**

- La corporación Discreet Guys Inc. necesita simular el funcionamiento de los ascensores de los nuevos edificios que se construirán.
- Las entradas se deben plantear para diferentes casos de prueba.
- Simulación:
    - Se necesita saber hacia qué oficina se dirige una persona.
    - La solución del problema debe tener un orden para el movimiento del ascensor.
    - Se necesita imprimir cada movimiento del ascensor y de las personas hasta que la ultima de ellas llegue a su oficina objetivo.
    - Se debe tener un apartado de consultas.

- La solución del problema debe implementar estructuras de datos genéricas.

**b) Recopilación de información:**

Con el objetivo de aprender cómo implementar estructuras de datos genéricas se realiza una búsqueda a fondo sobre cada estructura que se podría usar para el problema planteado.

"Generics add stability to your code by making more of your bugs detectable at compile time." … "This enhancement to the type system allows a type or method to operate on objects of various types while providing compile-time type safety. It adds compile-time type safety and eliminates the drudgery of casting." … "The net effect, especially in large programs, is improved readability and robustness."

Fuentes:

https://docs.oracle.com/javase/tutorial/extra/generics/index.html

https://www.youtube.com/

https://www.geeksforgeeks.org/

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.priorityqueue-2?view=net-6.0

https://www.tutorialspoint.com/

**c) Búsqueda de soluciones creativas:**

En este punto, nos planteamos mediante una lluvia de ideas las maneras de resolver el problema. Los resultados de la lluvia de ideas fueron:

a) Manejar la solución mediante Hashmap y priorityQueues.

b) Implementar una maxPriorityQueue, una minPriorityQueue, una linkedList y un hashmap genéricos.

**d) Transición de la formulación de ideas a los diseños preliminares:**

En este paso lo que haremos será evaluar las dos soluciones posibles a este problema planteado.

- Alternativa 1. Hashmap y priorityQueues:
  - Esta alternativa implica el uso de clases de java, por lo que el requerimiento de usar clases genéricas no se cumple.
  - En esta alternativa supone manejar las entradas con un piso de llegada y no con una oficina, lo que implica que uno de los requerimientos no se cumpla.

- Alternativa 2. MaxPriorityQueue, minPriorityQueue, linkedList y hashmap genéricos:
  - Esta alternativa supone una mayor complejidad a la hora de la implementación.

**e) Evaluación y Selección de la Mejor Solución:**

*Criterios:*

Para la selección de la solución tendremos en cuenta los siguientes criterios:

- Criterio A. Suplencia de Requerimientos:
  - [3] Cumple todos los requerimientos
  - [2] Incumple 1 solo requerimiento
  - [1] Incumple 2 o más requerimientos

- Criterio B. Complejidad de implementación:
  - [2] Poca complejidad
  - [1] Bastante complejidad

*Evaluación:*

| | Criterio A | Criterio B |
|---|---|---|
| Alternativa 1. Hashmap y priorityQueues | Incumple 2 o más requerimientos<br>1 | Poca complejidad<br>2 |
| Alternativa 2. MaxPriorityQueue, minPriorityQueue, linkedList y hashmap genéricos | Cumple todos los requerimientos<br>3 | Bastante complejidad<br>1 |

*Selección:* De acuerdo con la anterior evaluación debemos tomar la alternativa número 2 ya que obtuvo la mayor puntuación de acuerdo con los criterios establecidos.

**f) Preparación de Informes y Especificaciones:**

Especificación de Problema:

Problema: Simulación de ascensores

Entrada: Cantidad de edificios, identificador de cada edificio, personas que se encuentran en el edificio, cantidad de pisos y oficinas por piso.

Nombre de persona, piso en el que se encuentra y oficina a la que se dirige.

Salida: Movimientos de cada persona en cada edificio.

**7. Implementación del Diseño:**

Implementación en Java

Lista de Tareas para implementar:

a) Pedir las entradas, crear los objetos y ejecutar las acciones necesarias.

b) Buscar piso

Especificación de subrutinas:

a)

| | |
|---|---|
| Nombre: | elevatorBuilding |
| Descripción: | Método principal de la clase donde se ejecuta la mayor parte del programa |
| Entrada: | Cantidad de edificios, identificador de cada edificio, personas que se encuentran en el edificio, cantidad de pisos y oficinas por piso.<br>Nombre de persona, piso en el que se encuentra y oficina a la que se dirige. |
| Salida: | Movimientos de cada persona en cada edificio. |

b)

| | |
|---|---|
| Nombre: | searchFloorArrive |
| Descripción: | Calcula el piso hacia el que se dirige una persona |
| Entrada: | - numFloors : int, numero de pisos<br>- office : int, oficinas por piso<br>-officeB : int oficina hacia la que va una persona |
| Salida: | int : floor, piso hacia el que se dirige una persona. |

Construcción:

a)

```
public void elevatorBuilding() {
    Max_PriorityQueue<Integer>maxQ=new Max_PriorityQueue(10);
    Min_PriorityQueue<Integer>minQ=new Min_PriorityQueue(10);
    int numBuilding=sc.nextInt();
    for(int i=0;i<numBuilding;i++) {
        HashMap<String,Integer>person_office=new HashMap();
        HashMap<Integer,String>floorUp=new HashMap();
        HashMap<Integer,String>floorDown=new HashMap();
        String idBuilding=sc.next();
        int numPerson=sc.nextInt();
        int numFloors=sc.nextInt();
        int office=sc.nextInt();
        for(int j=0;j<numPerson;j++) {
            String name=sc.next();
            int actualfloor=sc.nextInt();
            int officeB=sc.nextInt();
            person_office.put(name,officeB);
            int arrivefloor=searchFloorArrive(numFloors,office,officeB);
            if(actualfloor<arrivefloor) {
                maxQ.add(officeB);
                floorUp.put(officeB, name);
            }else {
                minQ.add(officeB);
                floorDown.put(officeB, name);
            }
        }
        System.out.println();
        for(int j=0;j<person_office.size();j++) {
            System.out.println(person_office.keySet().get(j)+" se mueve a la oficina: "+person_office.values().get(j));
        }
        System.out.println();
        System.out.println("El orden de llegada es ");
        while(maxQ.peek()!=null) {
            for(int j=0;j<floorUp.size();j++) {
                if(floorUp.keySet().get(j).equals(maxQ.peek())) {
                    System.out.println(floorUp.get(maxQ.peek()));
                }
            }
            maxQ.poll();
        }
        while(minQ.peek()!=null) {
            for(int j=0;j<floorDown.size();j++) {
                if(floorDown.keySet().get(j).equals(minQ.peek())) {
                    System.out.println(floorDown.get(minQ.peek()));

                }
            }
            minQ.poll();
        }

    }
}
```

b)

```
public int searchFloorArrive(int numFloors,int office,int officeB) {
    int[][]building=new int[numFloors][office];
    int numberOffice=numFloors*office;
    int floor=0;
    for(int i=0;i<numFloors;i++) {
        for(int j=0;j<office;j++) {
            building[i][j]=numberOffice;
            numberOffice--;
        }
    }
    for(int i=0;i<numFloors;i++) {
        for(int j=0;j<office;j++) {
         if(building[i][j]==officeB) {
             floor=i+1;
         }

        }
    }
    return floor;
}
```