|  |  |
| --- | --- |
| **Integrantes** | |
| Esteban Dalel R | 201227078 |
| Pablo Alvarado Ceron | 201325754 |

# Estructura máxima 5 HOJAS

# Introducción

Para manejar los pedidos, la empresa requiere un sistema informático que controle los camiones, para encontrar comportamientos sospechosos, y las llamadas al Call Center, para ofrecer un mejor servicio y tomar las quejas de los clientes. Con estos requerimientos funcionales, se diseñó un programa en Java que controle la central de llamadas y los camiones, reportando eventos sospechosos.

# Derechos de propiedad

Nosotros Esteban Dalel R y Pablo Alvarado garantizamos que el presente trabajo y todos sus entregables son de nuestra propiedad o usados con los correspondientes permisos, sin incurrir en ningún tipo de falta de derechos de propiedad.

# Requerimientos Funcionales

Identifique los 3 requerimientos funcionales principales para el sistema propuesto y documéntelos siguiendo el siguiente formato:

|  |  |
| --- | --- |
| Nombre | ManejoInterfaz |
| Resumen | Se encarga de recibir como texto las órdenes del usuario y mostrarle, también como texto, las respuestas a sus requerimientos. |
| Entradas | |
| Órdenes en forma de texto (String) | |
| Resultados | |
| Respuesta en forma de texto (String) | |

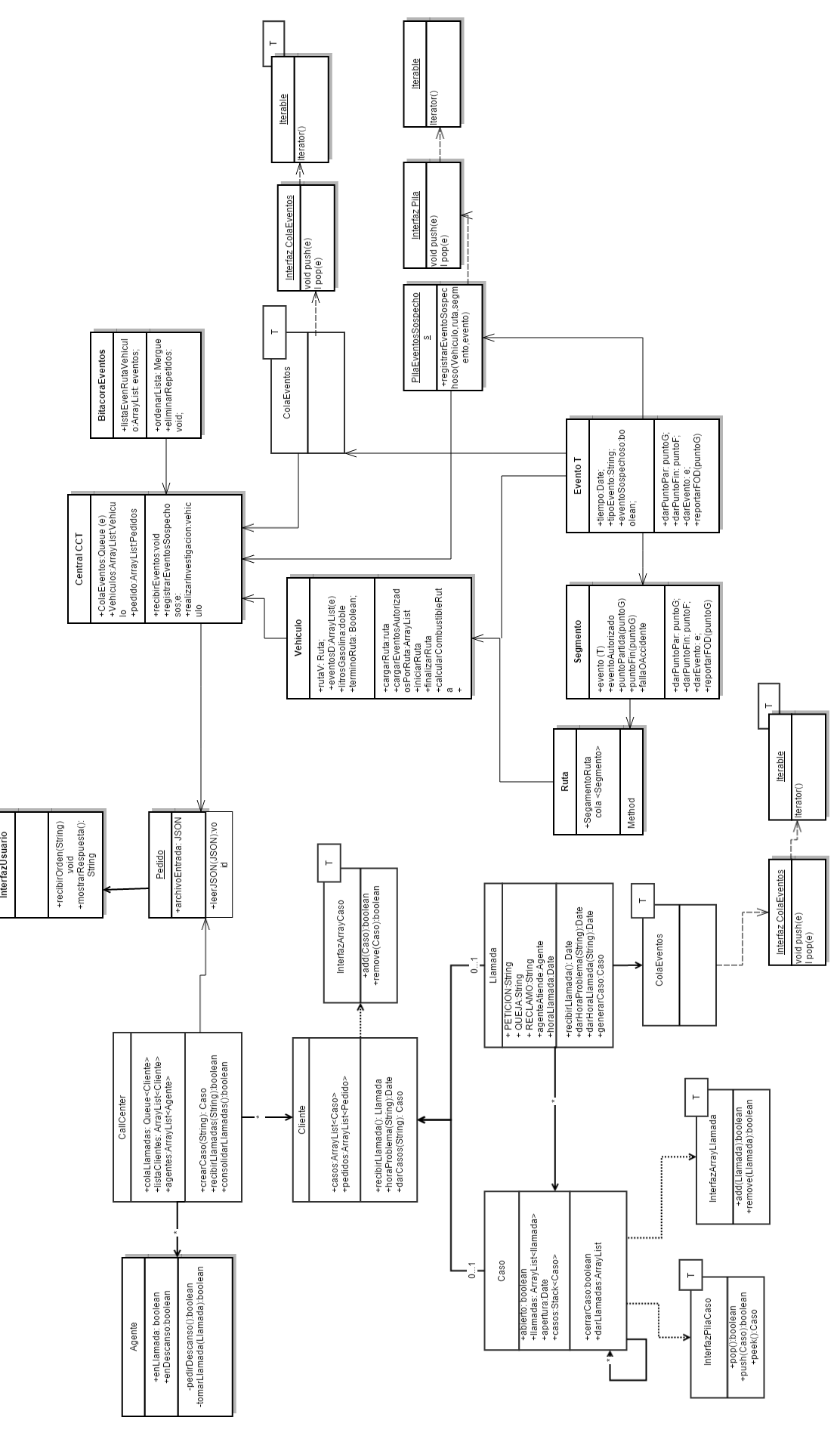
|  |  |
| --- | --- |
| Nombre | Analizar eventos registrados de un vehículo en una ruta |
| Resumen | Se deben analizar los eventos que se dan en una ruta, para los vehículos y según los parámetros dados; analizar si es un comportamiento sospechoso para merecer una investigación |
| Entradas | |
| Eventos ordenados por tiempo en el que suceden a lo largo de un segmento de una ruta.  Vehículo el cual tenía la ruta asignada | |
| Resultados | |
| Se envía el evento sospechoso a la pila de eventos sospechosos  En caso de que se dé un patrón, que amerite sospecha, se generara el informe | |

|  |  |
| --- | --- |
| Nombre | CallCenter |
| Resumen | Se encarga de recibir las llamadas, clasificarlas y atender los casos conforme se vayan reportando. |
| Entradas | |
| Llamadas (JSON) | |
| Resultados | |
| Respuesta en forma de texto (String) a ManejoInterfaz | |

# Modelo del mundo

Presente de forma clara su modelo del mundo usando la notación UML y explique los diferentes elementos del que consiste (para cada parte del proyecto). Indique como **las estructuras que usted creó,** las cuales deben expresarse en el modelo, satisfacen los requerimientos funcionales propuestos por el cliente al igual como los no funcionales.

Así mismo, indique cual fue el aporte de cada estudiante en el proyecto (quién hizo qué).



# Análisis de complejidad

De los requerimientos propuestos tome los 3 más importantes, copie el fragmento de código que satisface dicho requerimiento y haga un análisis de complejidad para cada uno. Indique el porqué de su respuesta, si piensa que no es la solución óptima explique porqué tomó dicha decisión y cual podría ser una mejor solución (si es la óptima indique porqué lo es).

Public boolean recibirLlamada(String datosCliente, String PQR, Date fechaProblema, String razonLlamada)

{

for(i=0;i<clientes.size;i++)

{

If(datosCliente==clientes[i])

{

Caso ultimaLlamada=new Caso(clientes[i], PQR, fechaProblema);

Return true;

}

}

Cliente esteCliente= new Cliente (datosCliente);

Clients.add(esteCliente);

Caso ultimaLlamada=new Caso(esteCliente, PQR, fechaProblema);

}

Recibe las llamadas, tomando los datos del cliente, el tipo de llamada que es, la fecha del problema, y la razón de la llamada. Recorre la lista de cliente buscando si este ya existe (ha llamado antes). Si lo encuentra, retorna verdadero, delegando la responsabilidad de tomar la llamada. Si no lo encuentra, crea el cliente y lo agrega a la lista de clientes y genera el objeto de la última llamada (actual) con los datos del cliente.

En el peor caso, deberá recorrer la lista completa (N) y generar los objetos (cte). En el mejor caso hallará el cliente en la punta de la lista (1) o estará vacía (0) y agregará el nuevo caso (cte). Por esto este método es de complejidad N.

Es la mejor solución posible, ya que no hay manera de saber en una lista sin orden si un objeto existe. Ahora bien, si clientes es una lista ordenada, la complejidad se reduce, ya que la búsqueda binaria es muy eficiente. Se puede usar heaps o hashTables como estructuras, optimizando la inserción y búsqueda.

|  |
| --- |
| public ArrayList eventosSospechosos(){  ArrayList sospechoso = new ArrayList;  for(int i=0; i<bitacoraEventos.size();i++){  EventoVehiculo temp = bitacoraEventos.get[i];  if(temp.esSospechoso()==true){  sospechoso.add(temp);  }  }  return sospechoso;  } |

De la bitácora de eventos, en la cual se tienen todos los eventos de todos los vehículos. Este método crea un array de los eventos sospechosos, de todos los vehículos. Su complejidad es N en el arreglo de eventos, porque ha de recorrerlo completo, revisando cada uno de ellos. Ya que agregar un evento requiere una sola operación iterativa, esta parte es cte. En el mejor de los casos ninguno es sospechoso, en el peor, todos los son y la operación generaría otro arreglo igual.

Esta es la única manera de hacer esta operación, ya que cada evento debe ser revisado.