Universidad Nacional Mayor de San Marcos

Universidad del Perú, decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela de Ingeniería de Software

Algorítmica II

Creación de software para empresa dedicada a la gestión en envío de paquetes en el Perú

Profesor:

Mg. Augusto Cortez Vásquez

Estudiantes:

Salinas Mejías, Ramsés Alfonzo 20200292

Ciudad Universitaria, Lima 2021

Resumen

El presente trabajo de exposición de proyecto expone la creación de un software el cual permite la gestión automatizada de una empresa que ofrece un servicio de entrega de paquetes, localizada en la ciudad de Lima, bajo el nombre de FEGPA (Facilitador en Gestión de Paquetes). Se contextualiza el proyecto en la actualidad, proporcionando antecedentes sobre la empresa en cuestión llamada Secufast, su problemática y los objetivos planteados a los que esta desea llegar. Adicionalmente, se investigan casos similares y se describen las soluciones planteadas en dichos casos. Con la finalidad de planificar una adecuada gestión del proyecto, se analizan los elementos descritos anteriormente en conjunto con los datos obtenidos de las encuestas de satisfacción realizadas por la empresa Secufast, donde sus clientes afirman que existe una inadecuada gestión del servicio realizado. Para esto se realizó un diagrama UML donde se especifican las clases del software creado en lenguaje Java. Cada una de las clases cumple funciones específicas que en conjunto implementan un sistema de gestión óptimo en todas sus fases, cumpliendo así con los requerimientos del servicio y de sus clientes.

Índice

Introducción

La comercialización es una actividad económica de la que nadie está exenta de estar involucrada, todos necesitamos de bienes para satisfacer nuestras necesidades y deseos. Estos mismos bienes no siempre son tan versátiles, portables o de fácil transporte; por ello, existen empresas que disponen, entre sus servicios, la alternativa de realizar el envío y entrega de bienes y productos.

Evidentemente empresas de este rubro también operan en el Perú, y a pesar de que la coyuntura actual ofrece un escenario favorable, el cual pretende traducirse en un aumento de la demanda frente a la restringida circulación de las personas por el riesgo de contagio, también puede ser contraproducente para la empresa si se ve envuelta en una incapacidad de proporcionar este servicio a una mayor escala de lo habitual y si persiste en seguir tratándola bajo el mismo esquema de proceso anterior, lo más probable es que se vea afectada su calidad. Una situación análoga es descrita por Garzón en 2013, respecto a la empresa Conserfast, la cual maneja procesos de recolección, empaque, entrega de paquetes, distribución, entre otros. Esta, tenía un sistema de organización para la información con respecto a las ventas y clientes en forma de libros, cuadernos y archivos Excel, los cuales generaban errores en la entrega y mayor tiempo para llevar a cabo el proceso.

Esta situación puede desencadenar una pérdida progresiva de clientes, propiciada por la insatisfacción experimentada frente a la calidad del servicio prestado. Para evitar esto, debemos revisar cuales son las fuentes de problemas potenciales en el proceso de entrega y envío de paquetes, las cuales analizaremos por medio de preguntas como: ¿Cómo se lleva a cabo el proceso de envió y entrega de productos?, ¿cuáles son sus etapas?, ¿qué parte del proceso no se está haciendo bien?, ¿qué se puede mejorar?, entre otras. Esto nos permitirá

diseñar y desarrollar un software con el fin de lidiar u optimizar ciertos puntos del proceso y por ende restaurar y/o elevar la calidad del servicio.

Capítulo 1: Antecedentes y Problemas

1.1 Antecedentes

Secufast, es una empresa dedicada a la distribución y transporte de bienes y productos empaquetados. Su sede principal se encuentra ubicada en Lima, en la provincia de Lima y el distrito de San Juan de Lurigancho y cuenta con otras sucursales en otros departamentos. Las formas con las que se brinda la atención a los clientes que solicitan sus servicios son de forma presencial o por medio de su página web. En este proceso, se pide al cliente sus datos personales, destino del paquete, remitente, destinatario (el nivel de especificación es variable), además de ciertas características destacables del paquete que se deben tomar en cuenta para realizar un adecuado transporte.

Para llevar el registro de datos necesarios que permitan la entrega del producto se cuenta con libros, cuadernos y hojas de Excel. La empresa cuenta con un sistema de clasificación de los paquetes de acuerdo a ciertas características del producto, como lo son la fragilidad, lugar de destino, vida útil (productos perecederos), entre otros. De acuerdo con esta clasificación son almacenados en contenedores. Estos mismos, posteriormente, son transportados por vehículos terrestres a su destino si este se encuentra en su dominio asignado, de lo contrario, se lleva a la sede central de la cual es derivada a una sucursal que se encuentre más cerca del destino final.

Al final se le brinda al cliente una encuesta en línea el cual tiene el fin de conocer la satisfacción del cliente con el estado de llegada de su producto, además de una sección de recomendaciones.

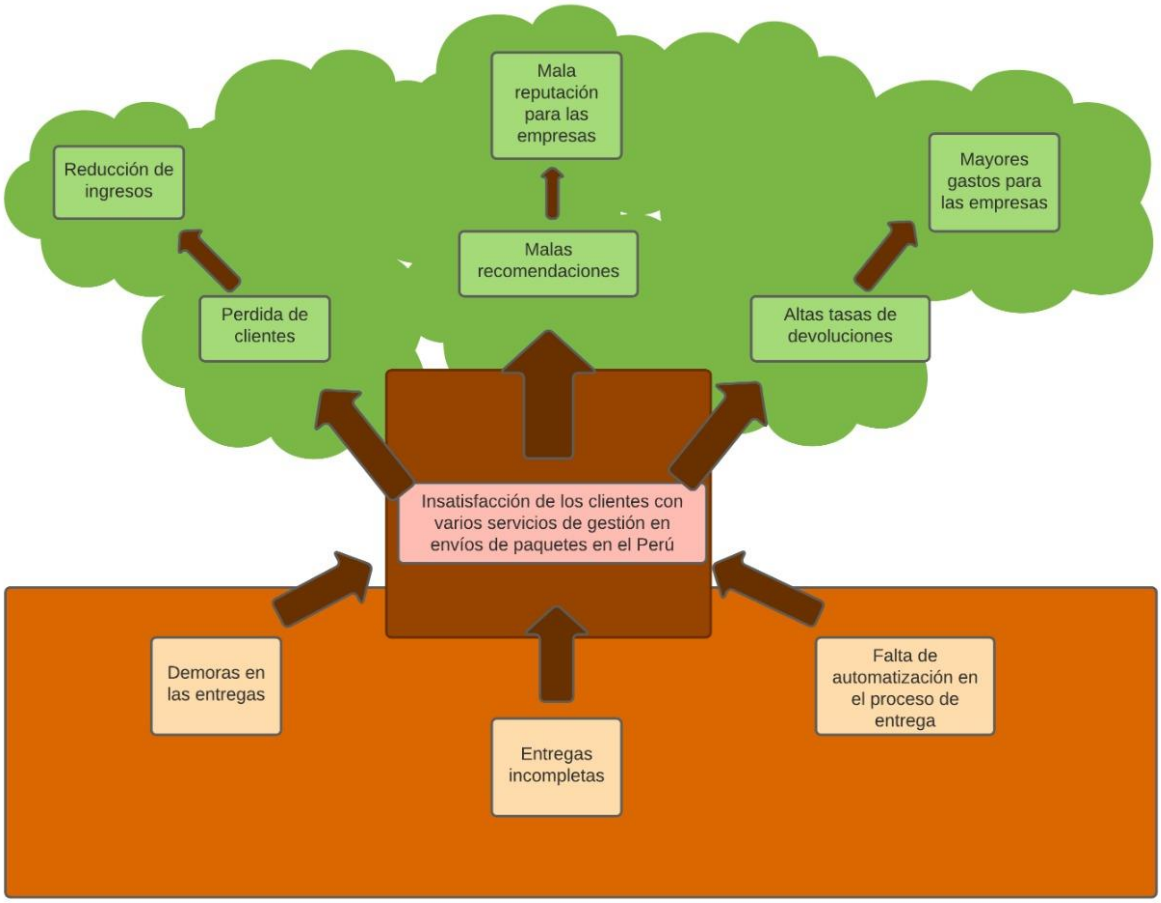
1.2. Problemática

Se realizó una revisión rápida de las encuestas recibidas de parte de los clientes donde se halló una calificación promedio de 4/10 en su grado de satisfacción con el servicio prestado. Por ello, se procedió a hacer una revisión más detallada de la sección de quejas y recomendaciones donde se encontró:

Para 10 clientes que solicitaron el servicio con destino a La Libertad, reportaron retrasos con respecto a la fecha u hora acordada de las entregas. Por otro lado, para los clientes de Callao se reportaron 24 quejas con respecto al estado en que se encontraba el paquete al momento de recibirlo. En Lima, se registraron 5 reclamos de los clientes, con respecto al contenido del paquete, solicitando una devolución y/o compensación. En Ica, se registró 9 reclamos con respecto a contenido incompleto del producto, según las declaraciones obtenidas, el producto venía en partes.

Para identificar las posibles fuentes, se realizó una encuesta interna a los integrantes de los distintos sectores que conforman la empresa para hallar las causas de la mala calidad del servicio. Esto permitió tomar en cuenta la cadena de envío de principio a fin. Con los testimonios obtenidos, se llega a la conclusión que la mayoría de los inconvenientes se manifestaban a causa de una falta de automatización en el proceso de entrega.

Gráfico 1. Árbol de problemas.



1. 2.1 Problema general.

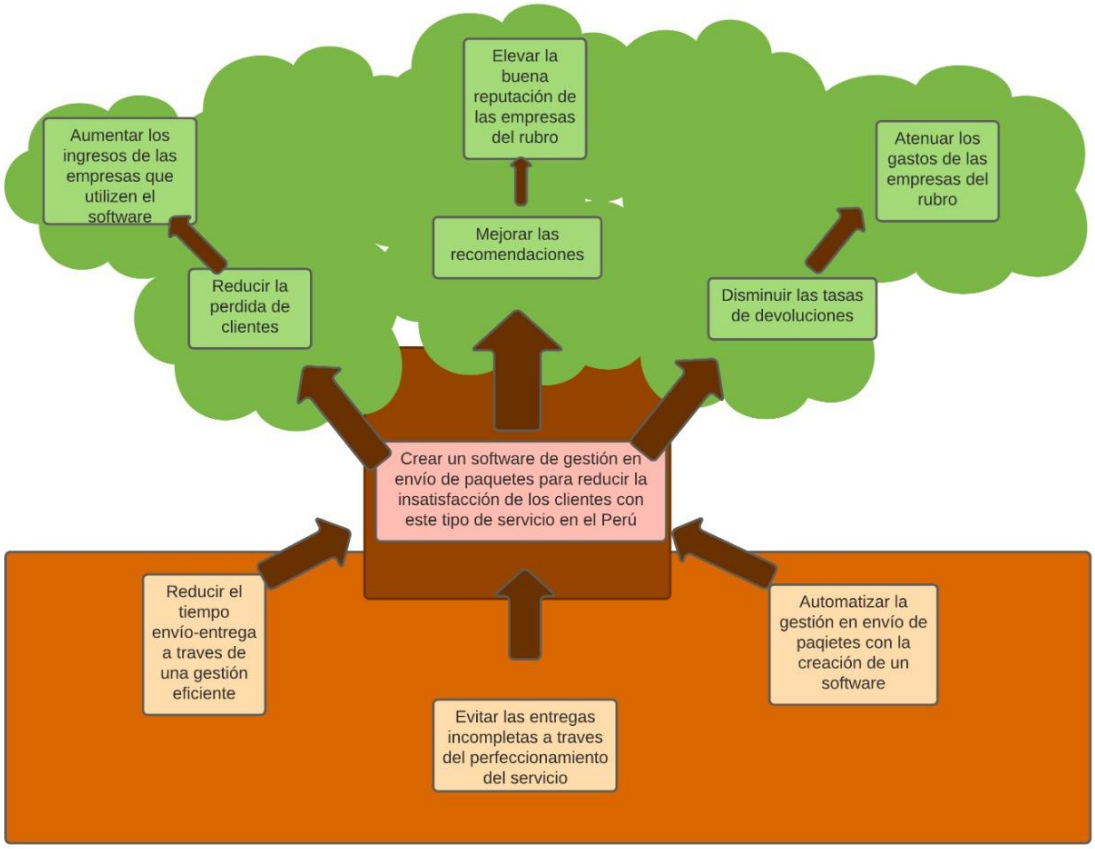
- Insatisfacción de los clientes con servicios de gestión en envíos de paquetes Secufast.

1.2.2 Problemas específicos:

- Demoras en las entregas de paquetes a los clientes.
- Entregas de paquetes incompletos.
- Falta de automatización dentro de la gestión en el proceso de entrega.

1.3. Objetivos

Gráfico 2. *Árbol de objetivos.*



1.3.1 Objetivo general:

- Crear un software de gestión en envío de paquetes para reducir la insatisfacción de los clientes con los servicios brindados por la empresa Secufast.

1.3.2 Objetivos específicos:

- Reducir el tiempo de envío - entrega a través de una gestión eficiente.
- Evitar las entregas incompletas a través del perfeccionamiento del servicio.
- Automatizar la gestión en envío de paquetes con la creación de un software.

1.4 Alcances

1.4.1 Alcance Geográfico:

El trabajo está desarrollado en el contexto nacional, orientado a solucionar los problemas de la empresa Secufast en lo que respecta a las situaciones descritas y registradas en las encuestas obtenidas de los clientes en el Perú.

1.4.2 Alcance Funcional:

El trabajo involucra el desarrollo de un programa con el objetivo de solucionar los problemas en lo que respecta la administración de la información pertinente de los productos con relación a sus usuarios y su etapa en el proceso de tal forma que se facilite la obtención de datos necesarios. En este programa, no se incluyen aspectos como el servicio de atención al cliente, utilidad del producto entre otros.

Capítulo 2 Marco Conceptual y Estado del Arte

2.1 Marco conceptual

2.1.1 Conceptos básicos

2.1.1.1 Envío

Es trasladar de forma física algo de un punto a otro. (Ecommerce Plataformas, 2021) Trasladar de un punto a otro paquetes, dinero, documentos o información; a través de diferentes medios.

2.1.1.2 Paquete

Lío o envoltorio bien dispuesto y no muy abultado de cosas de una misma o distinta clase (RAE,2021). Utilizado para trasladar su contenido, clasificar elementos, darle una protección adicional a su contenido, entre otras cosas.

2.1.1.3 Cliente

Persona que solicita o demanda un bien o servicio a una entidad capaz. (RAE, 2021). Dicha entidad asume una responsabilidad a cambia de una compensación.

2.1.1.4 Software

” Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora”.(RAE,2021)

2.1.1.5 Registro

Es un documento que almacena la respuesta u otro tipo de salida desde un proceso o actividad. Los registros son muestra de que una actividad tuvo sitio y podría

encontrarse en papel o en formato electrónico (Office of Goberné Commerce, 2009. Citado por Tapara, 2019.)

2.1.2 Conceptos Técnicos

2.1.2.1 Servicio de envío o Courier

Un servicio de Courier es una empresa que se dedica a las entregas especiales de paquetes, dinero, documentos o información. Uno de los estandartes principales del servicio Courier es que tienen tiempos de entrega más rápidos que cualquier otro método de transporte de documentos, por esto muchas empresas y personas confían en ellos. En otras palabras, los servicios de Courier tienen buena fama gracias a su velocidad superior y su capacidad de seguimiento, característica esencial que lo hace superior a los servicios de correo tradicionales. (AFE Logistics, 2021).

2.1.2.2 Sistemas de información gerencial

Según Domínguez (2012) citado por Tapara (2019), es un conjunto de sistemas de información que interactúan entre sí y que proporcionan información de suma importancia a la administración acerca de las necesidades que se tienen en, por ejemplo, las operaciones de la institución gerenciada.

2.1.2.3 Técnica de optimización (Cubicaje)

De acuerdo con lo mencionado por Jimenez et al. (2015), plantea situaciones en las que los contenedores donde se almacenan los productos a transportar no son adecuadamente aprovechadas, esto debido a factores como inadecuada colocación, combinación de peso, forma, volumen entre otros, lo cual genera más gastos en los costos del servicio de transporte.

3. Estado del arte

3.1 Diseño e implementación del sistema de administración y control de envíos o entregas de paquetes y documentos para la empresa Conserfast

Autor: Milton Raúl Garzón Iñiguez

Problema: La empresa Conserfast carece de un sistema automatizado de administración y control de la información, lo que genera perdida y mal manejo de esta. En consecuencia, generan errores en la entrega y mayor tiempo para llevar a cabo el proceso, lo que conlleva a una pérdida de clientes.

Objetivo Principal: Diseñar e implementar el sistema de administración y control de envíos o entrega de paquetes y documentos para la empresa Conserfast.

Metodología: Paradigma Espiral Incremental y la metodología OMT Rumbaugh.

3.2 Propuesta de implementación de un sistema de gestión Courier en la empresa Express Mail Servicie E.I.R.L. - Piura; 2021

Autor: María del Socorro Vergara Feijoo

Problema: La empresa maneja los procesos de forma manual, además que 60% de los trabajadores encuestados pertenecientes al área administrativa, no se encuentran satisfechos con el sistema manejado

Objetivo Principal: Proponer la Implementación de un Sistema de Gestión Courier en la empresa Express Mail Servicie E.I.R.L. - Piura; 2021, para mejorar la administración del envío y entrega a los clientes.

Metodología: Proceso Unificado Racional (RUP), ya que consideró que le brindaba las herramientas adecuadas y más utilizadas para el análisis, diseño, implementación y documentación de software orientado a objetos**.**

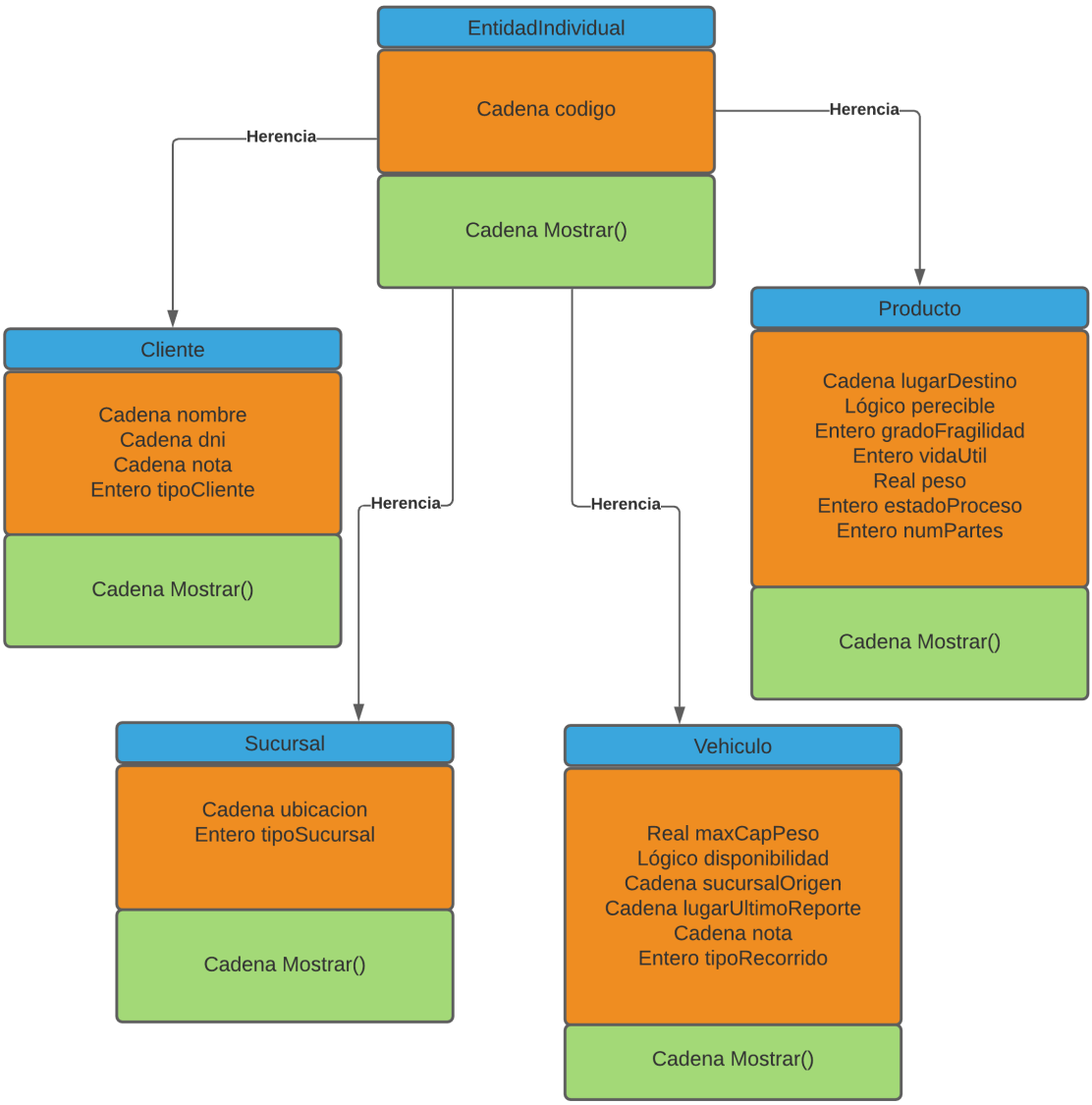
Resumen:

Para poder llevar a cabo la mencionada metodología RUP, se tuvo en cuenta dos fases. En la fase 1 (análisis) definió los requerimientos funcionales y no funcionales, resaltando que la finalidad del software era automatizar los procesos de la empresa, para tener un mejor control y de esta manera brindar un mejor servicio; a la misma vez detalló la visión del proyecto (mejorar el tiempo de entrega de envíos y reducir el índice de pérdidas de envíos realizadas a provincias, reduciendo los pagos realizados por penalidad de pérdida). En la fase 2 (diseño) definió los demás requerimientos y riesgos que podía encontrar, además de elaborar y mejorar los diagramas UML (Modelamiento de los Procesos del Sistema). Al finalizar logró su objetivo y validó su tesis, en buena medida (según su propia conclusión) a la recopilación de información que hizo con anterioridad, la misma que fue esencial para determinar los requerimientos y a partir de ahí construir su proyecto. (Vergara, 2021).

4. Especificación de la solución

4.1 Diagrama de clases

Familia de herencia: EntidadIndividual.



Familia de herencia: EntidadLista.

Agregación:

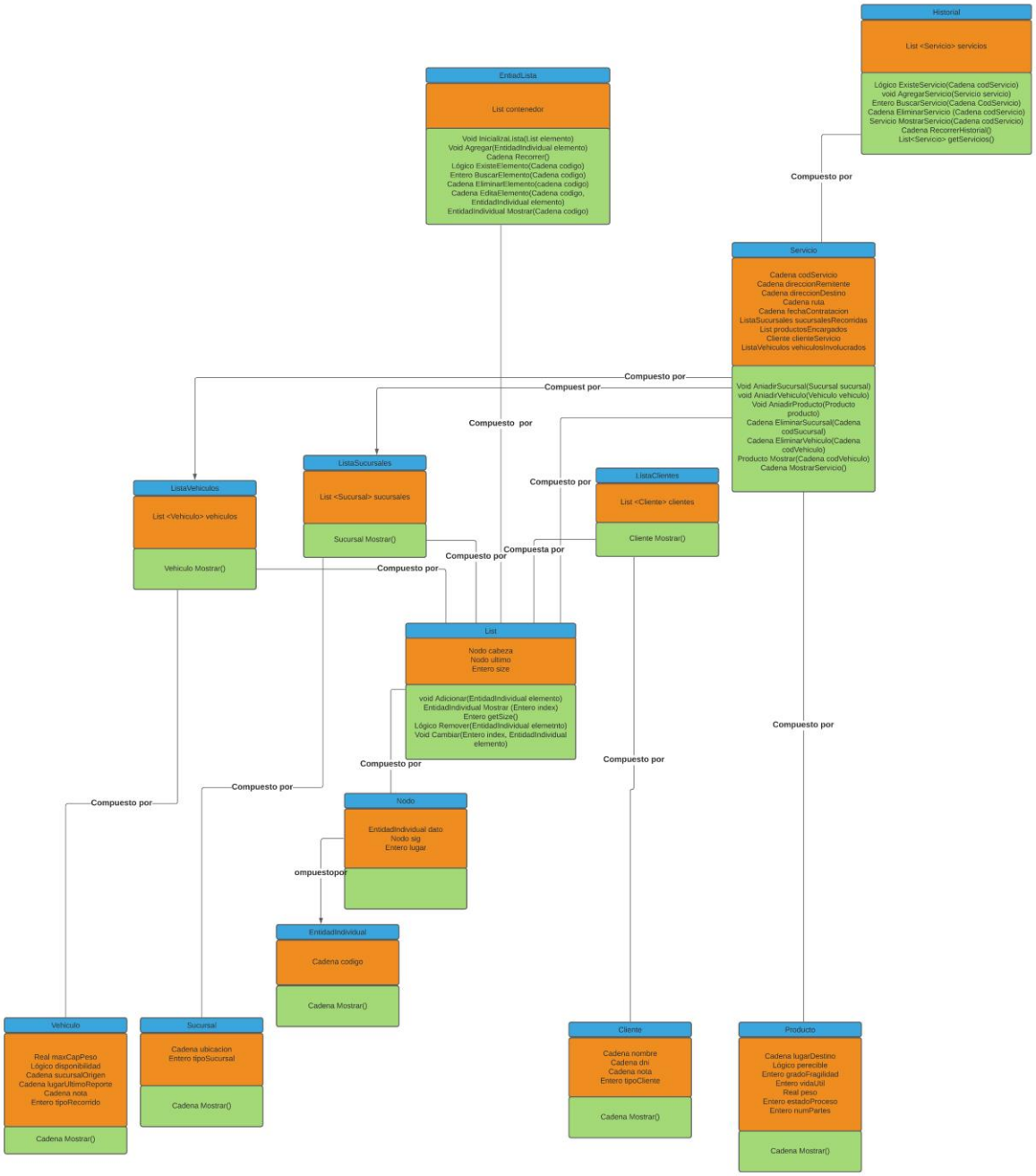
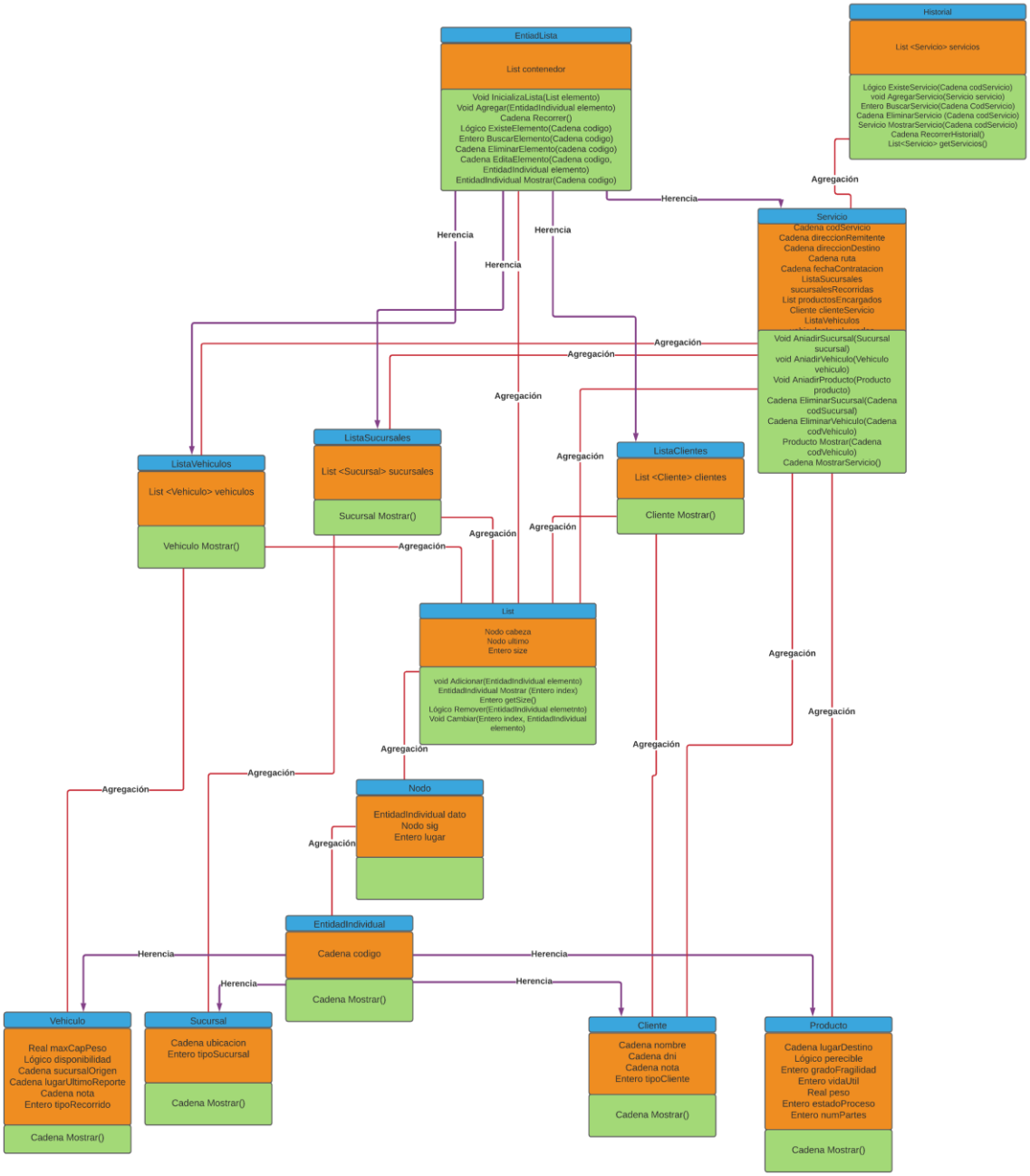

Historial
List <Servicio> servicios
Lógico ExisteServicio(Cadena codServicio)
void AgregarServicio(Servicio servicio)
Entero BuscarServicio(Cadena CodServicio)
Cadena EliminarServicio (Cadena codServicio)
Servicio MostrarServicio(Cadena codServicio)
Cadena RecorrerHistorial()
List<Servicio> getServicios()
Compuesto por
EntiadLista
List contenedor
Void InicializaLista(List elemento)
Void Agregar(EntidadIndividual elemento)
Cadena Recorrer()
Lógico ExisteElemento(Cadena codigo)
Entero BuscarElemento(Cadena codigo)
Cadena EliminarElemento(cadena codigo)
Cadena EditaElemento(Cadena codigo, EntidadIndividual elemento)
EntidadIndividual Mostrar(Cadena codigo)
Servicio
Cadena codServicio
Cadena direccionRemitente
Cadena direccionDestino
Cadena ruta
Cadena fechaContratacion
ListaSucursales sucursalesRecorridas
List productosEncargados
Cliente clienteServicio
ListaVehiculos vehiculosInvolucrados
Void AniadirSucursal(Sucursal sucursal)
void AniadirVehiculo(Vehiculo vehiculo)
Void AniadirProducto(Producto producto)
Cadena EliminarSucursal(Cadena codSucursal)
Cadena EliminarVehiculo(Cadena codVehiculo)
Producto Mostrar(Cadena codVehiculo)
Cadena MostrarServicio()
Compuesto por
Compuest por
Compuesto por
ListaSucursales
List <Sucursal> sucursales
Sucursal Mostrar()
ListaVehiculos
List <Vehiculo> vehiculos
Vehiculo Mostrar()
ListaClientes
List <Cliente> clientes
Cliente Mostrar()
Compuesto por
Compuesta por
Compuesto por
Compuesto por
List
Nodo cabeza
Nodo ultimo
Entero size
void Adicionar(EntidadIndividual elemento)
EntidadIndividual Mostrar (Entero index)
Entero getSize()
Lógico Remover(EntidadIndividual elemetnto)
Void Cambiar(Entero index, EntidadIndividual elemento)
Compuesto por
Compuesto por
Compuesto por
Compuesto por
Compuesto por
Nodo
EntidadIndividual dato
Nodo sig
Entero lugar
ompuestopor
EntidadIndividual
Cadena codigo
Cadena Mostrar()
Vehiculo
Real maxCapPeso
Lógico disponibilidad
Cadena sucursalOrigen
Cadena lugarUltimoReporte
Cadena nota
Entero tipoRecorrido
Cadena Mostrar()
Sucursal
Cadena ubicacion
Entero tipoSucursal
Cadena Mostrar()
Cliente
Cadena nombre
Cadena dni
Cadena nota
Entero tipoCliente
Cadena Mostrar()
Producto
Cadena lugarDestino
Lógico perecible
Entero gradoFragilidad
Entero vidaUtil
Real peso
Entero estadoProceso
Entero numPartes
Cadena Mostrar()

Diagrama completo:



4.2 Entidades

4.2.1 EntidadIndividual

TAD_Abstracta EntidadIndividual

Inicio

Datos:

Cadena codigo

Métodos:

EntidadIndividual(Cadena cod)

Mostrar() //Método abstracto

getCodigo()

Fin

4.2.2 Nodo

Tipo Nodo

EntidadIndividual dato

Nodo sig

Entero lugar

Fin

4.2.3 List

TAD List

Inicio

Datos:

Nodo cabeza

Nodo ultimo

Entero size

Métodos:

List()

Adiciona(EntidadIndividual elemento)

Mostrar(Entero index)

getSize()

Remover(EntidadIndividual elemento)

Cambiar(Entero index, EntidadIndividual elemento)

Fin

4.2.4 EntidadLista

TAD_Abstracta EntidadLista

Inicio

Datos:

contenedor : List tipo EntidadIndividual

Métodos:

InicializaLista(List elemento)

Agregar(EntidadIndividual elemento)

Recorrer()

ExisteElemento(Cadena cod)

BuscarElemento(Cadena cod)

EliminarElemento(Cadena cod)

EditaElemento(Cadena cod, EntidadIndividual elemento)

Mostrar() //Método abstracto

Fin

4.2.5 Cliente : EntidadIndividual

TAD Cliente : EntidadIndividual

Inicio

Datos:

Cadena nombre, dni, nota

Entero tipoCliente // 1: cliente nuevo; 2: cliente concurrente; 3: cliente VIP

Métodos:

Cliente(Cadena coCli, Cadena name, Cadena dni, Entero tipoCli, Cadena nota)

Mostrar ()

Fin

4.2.6 Producto : EntidadIndividual

TAD Producto : EntidadIndividual

Inicio

Datos:

Cadena lugarDestino

Lógico perecible //Verdadero: es predecible; Falso: no es perecible

Entero gradoFragilidad //1: fragilidad baja; 2: fragilidad media; 3: fragilidad alta

Entero vidaUtil // días

Real peso

Entero eProceso //1: solicitado; 2:Paquete en almacén 3:en transporte 4: entregado

Entero numPartes

Métodos:

Producto(Cadena codigoPro, Cadena lugarDest, Lógico pereci, Entero gradoFra, Entero vidaU, Real pso, Entero estadoProce, Entero numeroPartes)

Mostrar()

Fin

4.2.7 Vehículo : EntidadIndividual

TAD Vehículo : EntidadIndividual

Inicio

Datos:

Real maxCapPeso

Lógico disponibilidad // Verdadero:disponible; Falso: no disponible

Cadena sucursalOrigen, lugarUltimoReporte, nota

Entero tipoRecorrido //1: urbano; 2: interprovincial; 3: interregional

Métodos:

Vehículo(Real maxCapP, Lógico dispon,Cadena sucurOrigen, Cadena codVeh, Entero tipoRe, Cadena lugarUltiRepor, Cadena note)

Mostrar ()

Fin

4.2.8 Sucursal : EntidadIndividual

TAD Sucursal : EntidadIndividual

Inicio

Datos:

Cadena ubicación

Entero tipoSucursal //1: central nacional; 2: central regional; 3: central provincial; 4: central distrital

Métodos:

Sucursal(Cadena ubi, Cadena codSucur, Entero tipoSu)

Mostrar ()

Fin

4.2.9 ListaClientes : EntidadLista

TAD ListaClientes : EntidadLista

Inicio

Datos:

clientes: List tipo Cliente

Métodos:

ListaClientes()

Mostrar(Cadena codCli)

getClientes()

Fin

4.2.10 ListaVehiculos : EntidadLista

TAD ListaVehiculos : EntidadLista

Inicio

Datos:

vehiculos: List tipo Vehiculo

Métodos:

ListaVehiculos()

Mostrar(Cadena codVe)

getVehiculos()

Fin

4.2.11 ListaSucursales : EntidadLista

TAD ListaSucursales : EntidadLista

Inicio

Datos:

sucursales: List tipo Sucursal

Métodos:

ListaSucursales()

Mostrar(Cadena codSucur)

getSucursales()

Fin

4.2.12 Servicio : EntidadLista

TAD Servicio : EntidadLista

Inicio

Datos:

Cadena direccionRemitente, direccionDestino, ruta, codServicio, fechaContratacion, fechaCumplimiento

ListaSucursales sucursalesRecorridas

productosEncargados: List tipo Producto tipo Producto

Cliente clienteServicio

ListaVehiculos vehiculosInvolucrados

Métodos:

Servicio(Cadena dirRemitente, Cadena dirDestino, Cadena ruta, Cadena codService, Cliente cli, Cadena fechaContrata)

AniadirSucursal(Sucursal sucursal)

AniadirProducto(Producto producto)

AniadirVehiculo(Vehiculo vehiculo)

MostrarServicio()

EliminarSucursal(Cadena codSucursal)

EliminarVehiculo(Cadena codVehiculo)

Mostrar(codProducto)

setEstadoProcesoProductos(Entero estado)

Fin

4.2.13 Historial

TAD Historial

Inicio

Datos:

servicios: List tipo Servicio

Métodos:

AgregarServicio(Servicio service);

RecorrerHistorial()

EliminarServicio(Cadena codServicio)

ExisteServicios(Cadena codServicio)

BuscarServicios(Cadena codServicio)

MostrarServicio(Cadena codServicio)

Fin

4.3 Especificación e implementación.

4.3.1 EntidadIndividual

4.3.1.1 EntidadIndividual(Cadena cod)

Especificación:

Cadena codigo //Atributo de clase
Cadena cod //Parámetro del método

A: Precondición: {}
P: FUN EntidadIndividual (cod: Cadena)
B: Postcondición {codigo <- cod}

Implementación:

```
Acción EntidadIndividual (Cadena cod)
Inicio
        Codigo <- cod
Fin Acción
```

4.3.1.2 getCodigo()

Especificación:

Cadena codigo //Atributo de clase

A: Precondición: {}
P: FUN getCodigo() DEV (codigo: Cadena)
B: Postcondición { }

Implementación:

```
Cadena Acción getCadena ()
Inicio
        retornar codigo
Fin Acción
```

4.3.2 List

4.3.2.1 List()

Especificación:

Nodo cabeza //Atributo de clase
Nodo ultimo //Atributo de clase
Entero size //Atributo de clase
A: Precondición: {}
P: FUN List ()
B: Postcondición { }

Implementación:

```
Acción List()
Inicio
        cabeza <- nulo
        ultimo <- nulo
        size <- 0
Fin Acción
```

4.3.2.2 add (EntidadIndividual elemento)

Especificación:

```
Nodo cabeza                          //Atributo de clase
Nodo ultimo                          //Atributo de clase
Entero size                           //Atributo de clase

EntidadIndividual elemento           //Parámetro del método
Nodo n                               //salida
```

A: Precondición: {}
P: FUN add (elemento: ElementoIndividual)
B: Postcondición { size <- size + 1}

Implementación:

```
Acción add (ElementoIndividual elemento)
Inicio
        size <- size + 1
        n <- nuevo Nodo()
        n.dato <- elemento
        n.sig <- nulo
        n.lugar <- size
        Si (cabeza = nulo)
                cabeza <- n
        Sino
                ultimo <- n
        FinSi
        ultimo <- n
Fin Acción
```

4.3.2.3 Mostrar (Entero index)

Especificación:

```
Nodo cabeza                          //Atributo de clase
Entero size                          //Atributo de clase
Entero index                         //Parámetro del método
EntidadIndividual elemento           //salida
Nodo n
```

A: Precondición: $\{index \in N \cup \{0\} / index \leq size \}$
P: FUN Mostrar (index: Entero) DEV (elemento: ElementoIndividual)
B: Postcondición { Si index = n.lugar
retorna n.dato}

Implementación:

```
EntidadIndividual  Acción Mostrar (Entero index)
Declaración de Variables
        Nodo n
        EntidadIndividual elemento<- nulo
Inicio
        n<- cabeza
        Hacer
                Si n.lugar = index
                        elemeto<- n.dato
                Sino
                        n<-n.sig
                FinSi
        Mientras elemento = nulo
        retornar elemento
Fin Acción
```

4.3.2.4 getSize ()

Especificación:

Entero size //Atributo de clase

A: Precondición: {}
P: FUN getSize () DEV (size: Entero)
B: Postcondición { }

Implementación:

```
Entero Acción getSize ()
Inicio
        retornar size
Fin Acción
```

4.3.2.5 Remover (EntidadadIndividual elemento)

Especificación:

Nodo cabeza	//Atributo de clase
Nodo ultimo	//Atributo de clase
Entero size	//Atributo de clase
EntidadIndividual elemento	//Parámetro del método
Lógico eliminado	//salida
Nodo n	

A: Precondición: { }
P: FUN Remover(EntidadIndividual elemento) DEV (eliminar: Lógico)
B: Postcondición { eliminado= elemento ∈ List }

Implementación:

```
Lógico Acción Remover (EntidadIndividual elemento)
Declaración de Variables
        Nodo n
        Lógico eliminar
Inicio
        Si cabeza.dato = elemento entonces
                elimina<-VERDADERO
                Si size-1=0 entonces
                        cabeza<- nulo
                        ultimo<- nulo
                Sino
                        n<-cabeza.sig
                        cabeza<-n
                        cabeza.lugar<-cabeza.lugar-1
                        Si size-1=1 entonces
                                ultimo<-cabeza
                        Sino
                                Hacer
                                        n<- n.sig
                                        n.lugar<- n.lugar-1
                                Mientras size-1>n.lugar
                                ultimo<-n
                        FinSi
                FinSi
        Sino
                Si ultimo.dato = elemento
                        elimina<-VERDADERO
                        Si size-1=0
                                ultimo <- nulo
                                cabeza<- nulo
                        Sino
                                Si size-1=1
                                        ultimo<-cabeza
                                Sino
                                        n<- cabeza
                                        Hacer
                                                n <- n.sig
                                                n.lugar<- lugar-1
                                        Mientras n.lugar<size-1
                                        ultimo<-n
                                FinSi
                        FinSi
                Sino
                        n<- cabeza
                        Hacer
                                n<-n.sig
                                Si n = elemento entonces
                                        elimina<-VERDADERO
                                        Hacer
                                                n <- n.sig
                                                n.lugar<- lugar-1
                                        Mientras n.lugar<size-1
                                        ultimo<-n
                                FinSi
                        Mientras ultimo<>n
                FinSi
        FinSi
        Si eliminar=VERDADERO entonces
                size<- size-1
        FinSi
        retornar eliminar
Fin Acción
```

4.3.2.6 Cambiar(Entero index, EntidadIndividual elemento)

Especificación:

Entero size	//Atributo de clase
EntidadIndividual elemento	//Parámetro del método
Entero index	//Parámetro del método

Nodo n

A: Precondición: { $index \in N \cup \{0\} / index \leq size$ }
P: FUN Cambiar (index: Entero, elemento: ElementoIndividual)
B: Postcondición { Si n.lugar=index entonces n.dato<-elemento}

Implementación:

```
EntidadIndividual  Acción Mostrar (Entero index, EntidadIndividual elemento)
Declaración de Variables
        Nodo n
Inicio
        n<- cabeza
        Hacer
                Si n.lugar = index
                        n.dato<- n.elemento
                Sino
                        n<-n.sig
                FinSi
        Mientras n.lugar-1<>index
Fin Acción
```

4.3.3 EntidadLista

4.3.3.1 InicializaLista(List elemento)

Especificación:

contenedor: List tipo EntidadIndividual //Atributo de clase
elemento: List //Parámetro del método

A: Precondición: {}
P: FUN InicializaLista (elemento: List)
B: Postcondición { }

Implementación:

```
Acción InicializaLista(List elemento)
Inicio
        contenedor <- elemento
Fin Acción
```

4.3.3.2 Agregar(EntidadIndividual elemento)

Especificación:

contenedor: List tipo EntidadIndividual //Atributo de clase
elemento: EntidadIndividual //Parámetro del método

A: Precondición: {}
P: FUN Agregar (elemento: List)
B: Postcondición { }

Implementación:

```
Acción Agregar(List elemento)
Inicio
        contenedor.add(elemento)
Fin Acción
```

4.3.3.3 Recorrer()

Especificación:

contenedor: List tipo EntidadIndividual //Atributo de clase
Cadena mensaje

A: Precondición: {contenedor.getSize()≥1}
P: FUN Recorrer () DEV (mensaje: Cadena)
B: Postcondición {Para i desde 1 hasta contenedor.getSize()
mensaje<- contenedor.mostrar(i).Mostrar()}

Implementación:

```
Cadena Acción Recorrer ()
Declaración de Variables
        Cadena mensaje<-“ ”
Inicio
        Si contendor.getSize() >= 1 entonces
                Para i desde 1 hasta contenedor.getSize() hacer
                        mensaje<- mensaje + contenedor.mostrar(i).Mostrar()
                FinPara
        Sino
                mensaje<- “Lista vacia”
        FinSi
        Retornar mensaje
Fin Acción
```

4.3.3.4 ExisteElemento(Cadena cod)

Especificación:

contenedor: List tipo EntidadIndividual //Atributo de clase
Cadena cod //Parámetro del método
Lógico existe
Entero i

A: Precondición: { }
P: FUN ExisteElemento (cod: Cadena) DEV (existe: Lógico)
B: Postcondición { existe= $i \in 1,2 \ldots contenedor.getSize()$,cod ∈ contenedor.Mostrar(i).codigo}

Implementación:

```
Lógico Acción ExisteElemento (Cadena cod)
Declaración de Variables
        Entero i
        Lógico existe
Inicio
        i<-1
        existe<-FALSO
        Mientras i <= contenedor.getSize() Hacer
                Si contenedor.Mostrar(i).codigo=cod entonces
                        existe<- VERDADERO
                FinSi
        FinMientras
        retornar existe
Fin
```

4.3.3.4 BuscarElemento(Cadena cod)

Especificación:

contenedor: List tipo EntidadIndividual //Atributo de clase
Cadena cod //Parámetro del método
Entero index
Entero i

A: Precondición: { ExisteElemento(cod)=VERDADERO;cotenedor.getSize()>=1}
P: FUN BuscarElemento (cod: Cadena) DEV (index: Entero)
B: Postcondición { Si cod=contenedor.Mostrar(i).codigo entonces index<- i}

Implementación:

```
Entero Acción BuscarElemento (Cadena cod)
Declaración de Variables
        Entero i
        Entero index
Inicio
        Para i desde 1 hasta contenedor.getSize() hacer
                Si contenedor.Mostrar(i)=cod entonces
                        index<- i
                FinSi
        FinPara
        Retornar index
Fin
```

4.3.3.4 EliminarElemento(Cadena cod)

Especificación:

contenedor: List tipo EntidadIndividual //Atributo de clase
Cadena cod //Parámetro del método
Cadena mensaje

A: Precondición: {}
P: FUN EliminarElemento (cod: Cadena) DEV (mensaje: Cadena)
B: Postcondición { }

Implementación:

```
Cadena Acción EliminarElemento (Cadena cod)
Declaración de Variables
        Cadena mensaje
Inicio
        Si ExisteElemento(cod) entonces
                Si contenedor.Remover(contenedor.Mostrar(BuscarElemento(cod))) entonces
                        mensaje<- “Eliminado con éxito”
                Sino
                        mensaje<- “Error al eliminar”
                FinSi
        Sino
                mensaje<- “No existe elemento con el código digitado”
        FinSi
        retornar mensaje
Fin
```

4.3.3.4 EditaElemento(Cadena cod, EntidadIndividual elemento)

Especificación:

```
contenedor: List tipo EntidadIndividual          //Atributo de clase
Cadena cod                                       //Parámetro del método
EntidadIndividual elemento                       //Parámetro del método
Cadena mensaje
```

A: Precondición: {}
P: FUN EditarElemento (cod: Cadena, elemento: EntidadIndividual) DEV (mensaje: Cadena)
B: Postcondición { }

Implementación:

```
Cadena Acción EditarElemento (Cadena cod, EntidadIndividual elemento)
Declaración de Variables
        Cadena mensaje
Inicio
        Si ExisteElemento(cod) entonces
                contenedor.Cambiar(BuscarElemento(cod),elemento)
                mensaje<- “Los datos han sido eliminados”
        Sino
                mensaje<- “No existe el elemento buscado”
        FinSi
        retornar mensaje
Fin
```

4.3.4 Cliente : EntidadIndividual

4.3.4.1 Cliente(Cadena coCli, Cadena name, Cadena documento, Entero tipoCli, Cadena not)

Especificación:

```
Cadena codigo                   //Atributo de clase EntidadIndividual
Cadena nombre                    //Atributo de clase
Cadena dni                       //Atributo de clase
Cadena nota                      //Atributo de clase
Entero tipoCliente              //Atributo de clase
Cadena name                     //Parámetro del método
Cadena documento                // Parámetro del método
```

Cadena not // Parámetro del método
Entero tipoCli // Parámetro del método
Cadena coCli // Parámetro del método

A: Precondición: {}
P: FUN Cliente(coCli, name, documento, not: Cadena; tipoCli: Entero)
B: Postcondición { }

Implementación:

```
Acción Cliente(Cadena coCli, Cadena name, Cadena documento, Entero tipoCli, Cadena not)
Inicio
        codigo<- coCli
        nombre <- name
        dni <- documento
        tipoCliente <- tipoClie
        nota <-not
Fin Acción
```

4.3.4.2 Mostrar() *//Se reescribe el método abstracto heredado (Polimorfismo)*

Especificación:

Cadena nombre //Atributo de clase
Cadena dni //Atributo de clase
Cadena nota //Atributo de clase
Entero tipoCliente //Atributo de clase
Cadena mensaje

A: Precondición: {}
P: FUN Mostrar () DEV (mensaje: Cadena)
B: Postcondición { }

Implementación:

```
Entero Acción getSize ()
Declaración de Variables
        Cadena mensaje
Inicio
        mensaje= nombre + dni + nota +tipoCliente
        retornar mensaje
Fin Acción
```

4.3.5 Producto: EntidadIndividual

4.3.5.1 Producto(Cadena codigoPro, Cadena lugarDest, Lógico pereci, Entero gradoFra, Entero vidaU, Real pso, Entero estadoProce, Entero múmeroPartes)

Especificación:

```
Cadena codigo                   //Atributo de clase EntidadIndividual
Cadena lugarDestino             //Atributo de clase
Lógico perecible                //Atributo de clase
Entero gradoFragidad            //Atributo de clase
Entero vidaUtil                 //Atributo de clase
Entero peso                     //Atributo de clase
Entero eProceso                 //Atributo de clase
Entero numPartes                //Atributo de clase

Cadena codigoPro                //Parámetro del método
Cadena lugarDest                //Parámetro del método
Lógico pereci                   //Parámetro del método
Entero gradoFra                 //Parámetro del método
Entero vidaU                    //Parámetro del método
Entero pso                      //Parámetro del método
Entero estadoProce              //Parámetro del método
Entero numeroPartes             //Parámetro del método
```

A: Precondición: {}
P: FUN Producto(Cadena codigoPro, lugarDest; Lógico pereci; Entero gradoFra, vidaU; Entero estadoProce, múmeroPartes; Real pso)
B: Postcondición { }

Implementación:

```
Acción Producto(Cadena codigoPro, Cadena lugarDest, Lógico pereci, Entero gradoFra, Entero
vidaU, Real pso, Entero estadoProce, Entero múmeroPartes)
Inicio
        codigo <- codigoPro
        lugarDestino <- lugarDest
        perecible <- pereci
        gradoFragilidad <-gradoFra
        vidaUtil <- vidaU
        peso <- pso
        eProceso<- estadoProce
        numPartes <-numeroPartes
Fin Acción
```

4.3.5.2 Mostrar() *//Se reescribe la clase abstracta heredad (Polimorfismo)*

Especificación:

Cadena codigo	//Atributo de clase EntidadIndividual
Cadena lugarDestino	//Atributo de clase
Lógico perecible	//Atributo de clase
Entero gradoFragidad	//Atributo de clase
Entero vidaUtil	//Atributo de clase
Entero peso	//Atributo de clase
Entero estadoProceso	//Atributo de clase
Entero numPartes	//Atributo de clase

A: Precondición: {}
P: FUN Mostrar () DEV (mensaje: Cadena)
B: Postcondición { }

Implementación:

```
Entero Acción getSize ()
Declaración de Variables
        Cadena mensaje
Inicio
        mensaje= código+ lugarDestino + perecible  + gradoFragilidad + vidaUtil + peso +
                eProceso + numPartes
        retornar mensaje
Fin Acción
```

4.3.6 Sucursal: EntidadIndividual

4.3.6.1 Sucursal(Cadena ubi, Cadena codSucur, Entero tipoSu)

Especificación:

```
Cadena codigo                   //Atributo de clase EntidadIndividual
Cadena ubicación                //Atributo de clase
Entero tipoSursal               //Atributo de clase
Cadena codSucur                 //Parámetro del método
Entero tipoSu                   // Parámetro del método
Cadena ubi                      // Parámetro del método
```

A: Precondición: {}
P: FUN Sucursal(Cadena ubi, Cadena codSucur, Entero tipoSu)
B: Postcondición { }

Implementación:

```
Acción Sucursal(Cadena ubi, Cadena codSucur, Entero tipoSu)
Inicio
        codigo<- codSucur
        ubicacion <- ubi
        tipoSucursal <- tipoSu
Fin Acción
```

4.2.6.2 Mostrar() *//Se reescribe la clase abstracta heredad (Polimorfismo)*

Especificación:

Cadena codigo //Atributo de clase EntidadIndividual
Cadena ubicacion //Atributo de clase
Entero tipoSursal //Atributo de clase
Cadena mensaje

A: Precondición: {}
P: FUN Mostrar () DEV (mensaje: Cadena)
B: Postcondición { }

Implementación:

```
Entero Acción getSize ()
Declaración de Variables
        Cadena mensaje
Inicio
        mensaje= codigo + ubicacion + tipoSucursal
        retornar mensaje
Fin Acción
```

4.3.7 Vehiculo: EntidadIndividual

4.3.7.1 Vehículo(Real maxCapP, Lógico dispon, Cadena sucurOrigen, Cadena codVeh, Entero tipoRe, Cadena lugarUltiRepor, Cadena note)

Especificación:

Cadena codigo //Atributo de clase EntidadIndividual
Real maxCapPeso //Atributo de clase
Lógico disponibilidad //Atributo de clase
Cadena sucursalOrigen //Atributo de clase
Cadena lugarUltimoReporte //Atributo de clase

Cadena nota //Atributo de clase
Entero tipoRecorrido //Atributo de clase

Cadena codVeh //Parámetro del método
Real maxCapP //Parámetro del método
Lógico dispon //Parámetro del método
Cadena sucurOrigen //Parámetro del método
Cadena lugarUltiRepor //Parámetro del método
Cadena note //Parámetro del método
Entero tipoRe //Parámetro del método

A: Precondición: {}
P: FUN Vehiculo(Real maxCapP, Lógico dispon, Cadena sucurOrigen, Cadena codVeh, Entero tipoRe, Cadena lugarUltiRepor, Cadena note)
B: Postcondición { }
Implementación:

```
Acción Vehiculo(Real maxCapP, Lógico dispon, Cadena sucurOrigen, Cadena codVeh, Entero tipoRe, Cadena lugarUltiRepor, Cadena note)
Inicio
        codigo <- codigoVeh
        maxCapPeso <- maxCapP
        disponibilidad <- dispon
        sucursalOrigen <- sucurOrigen
        lugarUltimoReporte <- lugarUltiRepor
        nota <- note
        tipoRecorrido<- tipoRe
Fin Acción
```

4.3.5.2 Mostrar() *//Se reescribe la clase abstracta heredad (Polimorfismo)*

Especificación:

Cadena codigo //Atributo de clase EntidadIndividual
Real maxCapPeso //Atributo de clase
Lógico disponibilidad //Atributo de clase
Cadena sucursalOrigen //Atributo de clase
Cadena lugarUltimoReporte //Atributo de clase
Cadena nota //Atributo de clase
Entero tipoRecorrido //Atributo de clase

A: Precondición: {}
P: FUN Mostrar () DEV (mensaje: Cadena)
B: Postcondición { }

Implementación:

```
Entero Acción getSize ()
Declaración de Variables
        Cadena mensaje
Inicio
        mensaje= codigo+ maxCapPeso + disponibilidad  + sucursalOrigen +
        lugarUltimoReporte + tipoRecorrido + nota
        retornar mensaje
Fin Acción
```

4.3.8 ListaClientes : EntidadLista

4.3.8.1 ListaClientes()

Especificación:

clientes: List tipo Cliente //Atributo de clase
InicializaLista(List elemento) //Método de la clase EntidadLista

A: Precondición: {}
P: FUN ListaClientes ()
B: Postcondición { }

Implementación:

```
Acción ListaClientes()
Inicio
        InicializaLista(clientes)
Fin Acción
```

4.3.8.2 Mostrar(Caneda codCli)*//Se reescribe el método abstracto heredado (Polimorfismo)*

Especificación:

clientes : List tipo Cliente //Atributo de clase
Mostrar(Entero index) //Método de la clase List
BuscarElemento(Cadena codigo) //Método de la clase EntidadLista
ExisteElemento(Cadena codigo) //Método de la clase EntidadLista
Cadena codCli //Parámetro del método
Cliente cli

A: Precondición: {ExisteElemento(codCli)=VERDADERO}
P: FUN Mostrar (codCli: Cadena) DEV (cli: Cliente)
B: Postcondición { }

Implementación:

```
Cliente Acción Mostrar (codCli)
Declaración de Variables
        Cliente cli
Inicio
        cli<- clientes.Mostrar(BuscarElemento(codCli))
        retornar cli
Fin Acción
```

4.3.8.3 getClientes ()

Especificación:

clientes : List tipo Cliente //Atributo de clase

A: Precondición: {}
P: FUN getClientes () DEV (clientes: List tipo Cliente)
B: Postcondición { }

Implementación:

```
List<Cliente> Acción getClientes ()
Inicio
        retornar clientes
Fin Acción
```

4.3.9 ListaVehiculos : EntidadLista

4.3.9.1 ListaVehiculos()

Especificación:

vehiculos: List tipo Vehiculo //Atributo de clase
InicializaLista(List elemento) //Método de la clase EntidadLista

A: Precondición: {}
P: FUN ListaVehiculos ()
B: Postcondición { }

Implementación:

```
Acción ListaVehiculos()
Inicio
        InicializaLista(vehiculos)
Fin Acción
```

4.3.9.2 Mostrar(Caneda codVe) *//Se reescribe el método heredado (Polimorfismo)*

Especificación:

vehiculos : List tipo Vehiculo //Atributo de clase

Mostrar(Entero index) //Método de la clase List
BuscarElemento(Cadena codigo) //Método de la clase EntidadLista
ExisteElemento(Cadena codigo) //Método de la clase EntidadLista
Cadena codVe //Parámetro del método
Vehiculo veh

A: Precondición: {ExisteElemento(codVe)=VERDADERO}
P: FUN Mostrar (codVe: Cadena) DEV (veh: Vehiculos)
B: Postcondición { }

Implementación:

```
Vehiculo Acción Mostrar (codVe)
Declaración de Variables
        Vehiculo veh
Inicio
        veh<- vehiculos.Mostrar(BuscarElemento(codVe))
        retornar veh
Fin Acción
```

4.3.9.3 getVehiculos ()

Especificación:

vehiculos : List tipo Vehiculos //Atributo de clase

A: Precondición: {}
P: FUN getVehiculos () DEV (clientes: List tipo Vehiculo)
B: Postcondición { }

Implementación:

```
List<Vehiculo> Acción getVehiculos ()
Inicio
        retornar vehiculos
Fin Acción
```

4.3.10 ListaSucursales : EntidadLista

4.3.10.1 ListaSucursales()

Especificación:

sucursales: List tipo Sucursal //Atributo de clase
InicializaLista(List elemento) //Método de la clase EntidadLista

A: Precondición: {}
P: FUN ListaSucursales ()
B: Postcondición { }

Implementación:

```
Acción ListaSucursales()
Inicio
        InicializaLista(sucursales)
Fin Acción
```

4.3.10.2 Mostrar(Caneda codSucur) *//Se reescribe el método heredado (Polimorfismo)*

Especificación:

sucursales: List tipo Sucursal //Atributo de clase
Mostrar(Entero index) //Método de la clase List
BuscarElemento(Cadena codigo) //Método de la clase EntidadLista
ExisteElemento(Cadena codigo) //Método de la clase EntidadLista
Cadena codSucur//Parámetro del método
Sucursal sucur

A: Precondición: {ExisteElemento(codSucur)=VERDADERO}
P: FUN Mostrar (codSucur: Cadena) DEV (sucur: Sucursal)
B: Postcondición { }

Implementación:

```
Sucursal Acción Mostrar (codSucur)
Declaración de Variables
        Sucursal sucur
Inicio
        sucur<- sucursales.Mostrar(BuscarElemento(codSucur))
        retornar sucur
Fin Acción
```

4.3.10.3 getSucursales ()

Especificación:

sucursales : List tipo Sucursal //Atributo de clase

A: Precondición: {}
P: FUN getSucursales () DEV (sucursales: List tipo Sucursal)
B: Postcondición { }

Implementación:

```
List<Sucursal> Acción getSucursales ()
Inicio
        retornar sucursales
Fin Acción
```

4.3.11 Servicio : EntidadLista

4.3.11.1 Servicio(Cadena dirRemitente, Cadena dirDestino, Cadena ru, Cadena codService, Cliente cli, Cadena fechaContrata)

Especificación:

Cadena codServicio	//Atributo de clase
Cadena direccionRemitente	//Atributo de clase
Cadena direccionDestino	//Atributo de clase
Cadena ruta	//Atributo de clase
Cadena fechContratacion	//Atributo de clase
Cliente clienteServicio	//Atributo de clase
Cadena codService	//Parámetro del método
Cadena dirRemitente	//Parámetro del método
Cadena dirDestino	//Parámetro del método
Cadena ru	//Parámetro del método
Cadena fechContrata	//Parámetro del método
Cliente cli	//Parámetro del método
InicializaLista(List elemento)	//Parámetro del método

A: Precondición: {}
P: FUN Servicio(Cadena dirRemitente, Cadena dirDestino, Cadena ru, Cadena codService, Cliente cli, Cadena fechaContrata)
B: Postcondición { }

Implementación:

```
Acción Servicio(Cadena dirRemitente, Cadena dirDestino, Cadena ru, Cadena codService,
Cliente cli, Cadena fechaContrata)
Inicio
        codServicio <- codService
        direccionRemitente <- dirRemitente
        direccionDestino <- dirDestino
        ruta <- ru
        fechaContratacion <- fechaContrata
        clienteServicio <- cli
Fin Acción
```

4.3.11.2 AniadirSucursal(Sucursal sucursal)

Especificación:

sucursalesRecorridas: List tipo Sucursal //Atributo de clase
Sucursal sucursal //Parámetro del método

A: Precondición: {}
P: FUN AniadirSucursal(Sucursal sucursal)
B: Postcondición { }

Implementación:

```
Acción AniadirSucursal(Sucursal sucursal)
Inicio
        sucursalesRecorridas.add(sucursal)
Fin Acción
```

4.3.11.3 AniadirVehiculo(Vehiculo vehiculo)

Especificación:

vehiculosInvolucrados: List tipo vehiculo //Atributo de clase
Vehiculo vehiculo //Parámetro del método

A: Precondición: {}
P: FUN AniadirVehiculo(Vehiculo vehiculo)
B: Postcondición { }

Implementación:

```
Acción AniadirVehiculo(Vehiculo vehiculo)
Inicio
        vehiculosInvolucrados.add(vehiculo)
Fin Acción
```

4.3.11.4 AniadirProdcuto(Producto productos)

Especificación:

productosEncargados: List tipo Producto //Atributo de clase
Producto producto //Parámetro del método

A: Precondición: {}
P: FUN AniadirProdcuto(Producto productos)
B: Postcondición { }

Implementación:

```
Acción AniadirProdcuto(Producto productos)
Inicio
        sucursalesEncargados.add(elemento)
Fin Acción
```

4.3.11.5 MostrarServicio()

Especificación:

```
Cadena codServicio                  //Atributo de clase
Cadena direccionRemitente           //Atributo de clase
Cadena direccionDestino             //Atributo de clase
Cadena ruta                         //Atributo de clase
Cadena fechContratacion             //Atributo de clase
Cliente clienteServicio             //Atributo de clase
```

A: Precondición: { }
P: FUN MostrarServicio () DEV (mensaje: Cadena)
B: Postcondición { }

Implementación:

```
Cadena Acción MostrarServicio (codVe)
Inicio
        mensaje<- codServicio + direccionRemitente + direccionDestino + ruta + fechaContratacion
                + clienteServicio
        retornar mensaje
Fin Acción
```

4.3.11.6 EliminarSucursal(Cadena codSucursal)

Especificación:

```
ListaSucursales sucursalesRecorridas    //Atributo de clase
Cadena codSucursal                      //Parámetro del método
Cadena mensaje
```

A: Precondición: {}
P: FUN EliminarElemento (codSucursal: Cadena) DEV (mensaje: Cadena)
B: Postcondición { }

Implementación:

```
Cadena Acción EliminarSucursal (Cadena codSucursal)
Inicio
        Si sucursalesRecorridas.ExisteElemento(codSucursal) entonces
                Si sucursalesRecorridas.Remover(sucursalesRecorridas.Mostrar(sucursalesRecorridas.BuscarElemento(codSucursal))) entonces
                        mensaje<- “Eliminado con éxito”
                Sino
                        mensaje<- “Error al eliminar”
                FinSi
        Sino
                mensaje<- “No existe ssucursal con el código digitado”
        FinSi
        retornar mensaje
Fin
```

4.3.11.7 EliminarVehiculo(Cadena codVehiculo)

Especificación:

ListaVehiculos vehiculosInvolucrados //Atributo de clase
Cadena codEVehiculo //Parámetro del método
Cadena mensaje

A: Precondición: {}
P: FUN EliminarElemento (codVehiculo: Cadena) DEV (mensaje: Cadena)
B: Postcondición { }

Implementación:

```
Cadena Acción EliminarSucursal (Cadena codVehiculo)
Inicio
        Si vehiculosInvolucrados.ExisteElemento(codVehiculo) entonces
                Si vehiculosInvolucrados.Remover(vehiculosInvolucrados.Mostrar(vehiculosInvolucrados.BuscarElemento(codVehiculo))) entonces
                        mensaje<- “Eliminado con éxito”
                Sino
                        mensaje<- “Error al eliminar”
                FinSi
        Sino
                mensaje<- “No existe ssucursal con el código digitado”
        FinSi
        retornar mensaje
Fin
```

4.3.11.8 Mostrar(Caneda codProduto) *//Se reescribe el método heredado (Polimorfismo)*

Especificación:

productosEncargados: List tipo producto //Atributo de clase
Mostrar(Entero index) //Método de la clase List
BuscarElemento(Cadena codigo) //Método de la clase EntidadLista
ExisteElemento(Cadena codigo) //Método de la clase EntidadLista
Cadena codProducto //Parámetro del método
Producto pro

A: Precondición: {ExisteElemento(codProducto)=VERDADERO}
P: FUN Mostrar (codProducto: Cadena) DEV (pro: Producto)
B: Postcondición { }

Implementación:

```
Producto Acción Mostrar (codProducto)
Inicio
        pro<- productosEncargados.Mostrar(BuscarElemento(codProducto))
        retornar pro
Fin Acción
```

4.3.12 Historial

TAD Historial

Inicio

Datos:

servicios: List tipo Servicio

Métodos:

AgregarServicio(Servicio service);

RecorrerHistorial()

EliminarServicio(Cadena codServicio)

ExisteServicios(Cadena codServicio)

BuscarServicios(Cadena codServicio)

MostrarServicio(Cadena codServicio)

Fin

4.3.12.1 AgregarServicio(Servicio service)

Especificación:

servicios: List tipo Servicio //Atributo de clase
elemento: Servicio //Parámetro del método

A: Precondición: {}
P: FUN AgregarServicio (Servicio service)
B: Postcondición { }

Implementación:

```
Acción AgregarServicios(Servicio service)
Inicio
        servicios.add(elemento)
Fin Acción
```

4.3.12.2 RecorrerHistorial()

Especificación:

servicios: List tipo Servicio //Atributo de clase
Cadena mensaje

A: Precondición: {servicios.getSize()≥1}
P: FUN RecorrerHistorial () DEV (mensaje: Cadena)
B: Postcondición {Para i desde 1 hasta contenedor.getSize()
mensaje<- contenedor.mostrar(i).Mostrar()}

Implementación:

```
Cadena Acción RecorrerHistorial ()
Inicio
        Si servicios.getSize() >= 1 entonces
                Para i desde 1 hasta servicios.getSize() hacer
                        mensaje<- mensaje + servicios.mostrar(i).Mostrar()
                FinPara
        Sino
                mensaje<- “Lista vacia”
        FinSi
        Retornar mensaje
Fin Acción
```

4.3.12.3 ExisteServicio(Cadena codServicio)

Especificación:

servicios: List tipo Servicio //Atributo de clase
Cadena codServicio //Parámetro del método
Lógico existe
Entero i

A: Precondición: { }
P: FUN ExisteServico(codServicio: Cadena) DEV (existe: Lógico)
B: Postcondición { existe= $i \in 1,2 \ldots servicios.getSize()$,cod ∈ servicios.Mostrar(i).codigo}

Implementación:

```
Lógico Acción ExisteServicio (Cadena codServicio)
Inicio
        i<-1
        existe<-FALSO
        Mientras i <= servicios.getSize() Hacer
                Si servicios.Mostrar(i).codigo=cod entonces
                        existe<- VERDADERO
                FinSi
        FinMientras
        retornar existe
Fin
```

4.3.12.4 BuscarServicio(Cadena codServicio)

Especificación:

servicios: List tipo Servicio //Atributo de clase
Cadena codServicio //Parámetro del método
Entero index
Entero i

A: Precondición: { ExisteElemento(cod)=VERDADERO;cotenedor.getSize()>=1}
P: FUN BuscarServicio (codServicio: Cadena) DEV (index: Entero)
B: Postcondición { Si codServicio=servicios.Mostrar(i).codigo entonces index<- i}

Implementación:

```
Entero Acción BuscarServicio (Cadena codServicios)
Inicio
        Para i desde 1 hasta servicios.getSize() hacer
                Si servicios.Mostrar(i)=codServicios entonces
                        index<- i
                FinSi
        FinPara
        Retornar index
Fin
```

4.3.12.5 EliminarServicio(Cadena codServicio)

Especificación:

servicios: List tipo Servicio //Atributo de clase
Cadena codServicio //Parámetro del método
Cadena mensaje

A: Precondición: {}
P: FUN EliminarServicio (codServicio: Cadena) DEV (mensaje: Cadena)
B: Postcondición { }

Implementación:

```
Cadena Acción EliminarServicio (Cadena codServicio)
Inicio
        Si ExisteServicio(codServicio) entonces
                Si servicios.Remover(servicios.Mostrar(BuscarServicio(codSucursal))) entonces
                        mensaje<- “Eliminado con éxito”
                Sino
                        mensaje<- “Error al eliminar”
                FinSi
        Sino
                mensaje<- “No existe servicio con el código digitado”
        FinSi
        retornar mensaje
Fin
```

Conclusión

A partir del análisis de los puntos planteados dentro de la investigación, podemos concluir que la gestión de esta fase de comercialización mediante el software antes mencionado se ha dado de manera satisfactoria, reduciendo el tiempo de gestión y automatizando los procesos debido a las funciones implementadas. Los puntos analizados son, en primer lugar, la problemática, la cual data de manera global una falta de automatización en el proceso de entrega, específicamente por demoras, paquetes incompletos y deficiente gestión. En segundo lugar, los objetivos planteados, los cuales proponen la reducción de insatisfacción mediante la elaboración de un software que gestione de manera automatizada el servicio de entrega. Podemos añadir, los diseños elaborados por los autores Garzón y Vergara, los cuales se usaron como base en conjunto con el marco teórico presentado para la elaboración del software expuesto en este informe. Por último, las funciones expuestas, tales como la clasificación según la sucursal, el orden de pedido, el lugar de entrega y el seguimiento del estado del paquete, tal como si este está en camino, si está en almacén o si ya ha sido entregado. Además, guarda cada historial en un archivo Excel donde se mantienen los datos seguros y de fácil acceso. Por último, respondiendo las interrogantes planteadas en la introducción, el proceso de envío se debe llevar de manera secuencial, donde se analice cada etapa del flujo de gestión, las cuales son recepción del envío, empaquetado, movilidad y envío directo o indirecto.

Referencias Bibliográficas

AFE Logistics.(4 de mayo de 2021). *¿Qué es un Courier y cómo funciona?* https://afelogistics.com/blog/definicion-funciones-courier/

Ecommerce Platforms. (2021). ¿Qué es el envío? ¿Qué significa envío? https://ecommerce-platforms.com/es/glossary/shipping?ep_lang=es

Garzón Iñiguez, M. R. (2013). *Diseño e implementación del sistema de administración y control de envíos o entregas de paquetes y documentos para la empresa Conserfast* (Tesis para optar al grado de bachiller, QUITO/EPN/2013). http://bibdigital.epn.edu.ec/handle/15000/6013

Gil Vasquez, S. F.; Llave Moreno, M. A. y Munive Casallo, J. C.(2016). *Aplicación de modelo de programación dinámica para la asignación de recursos del área de fuerza de ventas de la empresa Total Potentials*. [Trabajo para optar por el título de licenciado, Universidad Peruana de Ciencias Aplicadas] https://repositorioacademico.upc.edu.pe/handle/10757/621494

Jimenez Castillo, J. , Bueno Solano, A. ,Jimenez Sanchez y Cedillo Campos, G. (2015). Cubicaje y su efecto económico en el costo logístico del transporte y competitividad empresarial. Instituto Mexicano del Transporte. https://www.imt.mx/archivos/Publicaciones/PublicacionTecnica/pt440.pdf

Tapara Ñañez, H. D. (2019). *Sistema web para automatizar el proceso de registro de incidencias del centro de gestión y control del metropolitano, 2019* (Tesis de licenciatura, Universidad privada Telesup) https://repositorio.utelesup.edu.pe/handle/UTELESUP/990

Vergara Feijoó, M. (2021). *Propuesta de implementación de un sistema de gestión Courier en la empresa Express Mail Service E.I.R.L. - Piura; 2021* (Tesis de licenciatura,

Universidad Católica de Los Ángeles). http://repositorio.uladech.edu.pe/handle/123456789/24252

http://repositorio.utmachala.edu.ec/bitstream/48000/7599/1/TCUAIC_2016_ISIST_CD0012.pdf