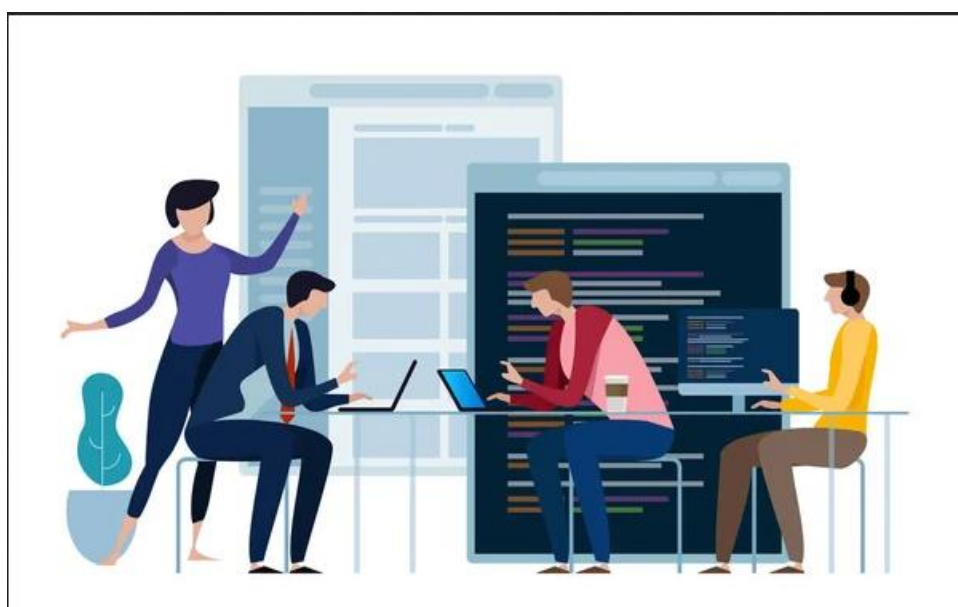


METODOLOGÍAS DE PROGRAMACIÓN II
INFORME TRABAJO INTEGRADOR
SISTEMA DE TRANSPORTE DE PASAJEROS



PROFESORA: CLAUDIA CAPPELLETTI

ALUMNOS: CARDOSO, MAURO

PRIMER CUATRIMESTRE

AÑO 2022

ÍNDICE

[1 - INTRODUCCIÓN](#)

[2 - DESCRIPCIÓN DEL PROBLEMA](#)

[3 - IMPLEMENTACIÓN](#)

[4 - METODOLOGÍAS DE TRABAJO](#)

[5 - FRAMEWORKS](#)

[6 - USO DE PATRONES](#)

[7 - ¿QUÉ REPOSITARIOS SE PODRÍAN UTILIZAR?](#)

[8 - ANEXOS](#)

INTRODUCCIÓN

El presente informe se realizó como “trabajo integrador” de la materia Metodologías de programación II, dictado por la Universidad Nacional Arturo Jauretche y a cargo de la profesora Cappelletti Claudia. El objetivo del mismo es la implementación de un sistema elegido por los alumnos, que plantee una problemática y su resolución aplicando las herramientas adquiridas durante la cursada de la materia. Todo realizado sobre “Dolphin Smalltalk professional v 7.0.5”.

El sistema que se optó por implementar es el de una agencia que brinda el servicio de transporte de pasajeros (remises). La implementación consta de 8 (ocho) clases, las cuales son: remisería, persona con sus subclases (cliente, chofer, empleado), vehículos, viajes y ubicación ([ANEXO 3](#)). También se realizaron los diagramas de clases y diagrama de secuencia que son la documentación que respalda el sistema elegido.

Algunas de las herramientas que se van a poder observar van a ser el uso de diccionarios y el uso de funciones como “Sellect, Collect, Reject y Detect”. Uso y ordenamiento de colecciones con funciones como “Sortedcollection”, como así también “Ocurrences of” ([ANEXO 4](#)).

A través de un menú con las principales opciones del sistema se podrá navegar por todo lo realizado, de manera ordenada y muy simple de entender. La idea principal del sistema es facilitar el trabajo de la agencia, realizando búsquedas de datos relevantes, iniciar viajes o finalizarlos, todo el manejo de su información en cuanto a guardado, eliminación, orden y todo lo que entendemos importante para el correcto funcionamiento del sistema y de negocio de la empresa en general.

Mediante la utilización de metodologías ágiles durante toda la cursada, fuimos realizando entregas y avances semanales. Utilizamos control de versiones para poder trabajar en equipo sin inconvenientes de trabajar con versiones anteriores o con errores ya solucionados por otro integrante del equipo. Lo más importante en el desarrollo del sistema fue la constante comunicación, la cual no solo fue entre los integrantes del equipo de desarrollo, sino también con la profesora a cargo de la cursada que actuó cual si fuera el cliente que daba los requerimientos.

A continuación, vamos a explicar con más detalle la implementación del sistema, con imágenes adjuntas para su mayor comprensión, y vamos a tratar, que, a través de la lectura del presente, quede bien clara la implementación y su documentación.

DESCRIPCIÓN DEL PROBLEMA

Como ya hemos mencionado, el presente informe describe en detalle la implementación de un sistema de “transporte de pasajeros” (remises). Esta labor realizada se debe a la solicitud por parte de la materia “Metodologías de Programación II” de la carrera de Ingeniería en Informática, de la Universidad Nacional Arturo Jauretche.

Como primera medida tuvimos que plantear una problemática de un negocio, por lo cual optamos por elegir una agencia de transporte de pasajeros. Se diseñó un diagrama de clases, un diagrama de secuencia y se lo implementó sobre “Dolphin Smalltalk professional v 7.0.5”. Una vez resuelto el diseño y sus respectivas clases y relaciones, aplicamos el diseño en la resolución de las problemáticas típicas de una agencia de este tipo. Algunas de estas problemáticas que se plantearon fueron las siguientes:

Se desea diseñar un sistema para una remisería, el mismo deberá poder registrar la siguiente información:

- Personal que trabaja en la remisería: nombres, apellido, DNI, teléfonos, direcciones, mail, horario y sueldo.
- Choferes: nombres, apellido, DNI, teléfonos, dirección, mail, horario, vehículos, horario y comisión.
- Clientes: nombres, apellido, DNI, teléfonos, dirección y mail.
- Vehículos: marca, modelo, año de fabricación y patente.

Además, se necesita que el sistema guarde información sobre los viajes como que cliente pidió el Remis, que chofer lo llevó a destino, que vehículo usó y las direcciones de origen del viaje y destino.

Luego del diseño del sistema, pusimos manos a la obra a la implementación de un sistema genérico, que puede satisfacer las necesidades básicas que hemos mencionado, de cualquier agencia de transportes, abstrayéndose de entrar en singularidades. De esta manera, construimos desde el diseño hasta la implementación y posterior presentación de la documentación, de un sistema íntegro, confiable y seguro, que cubre con las funcionalidades de una empresa de transporte de pasajeros. A continuación, entraremos en más detalle acerca de dicha implementación.

IMPLEMENTACIÓN

Para la implementación del sistema, entre otras funcionalidades, planteamos la problemática de iniciar un viaje, con todo lo que ello conlleva (clientes, vehículos, choferes, direcciones, importes, etc.) y su finalización una vez que el usuario que realizó el mismo haya llegado a destino.

```
"BIENVENIDOS A LA REMISERIA "CARDOSO-SHIMABUKURO"

INGRESE LA OPCION QUE DESEA

0 - AGREGAR EMPLEADO, CHOFER, VEHICULO O CLIENTE
1 - ASOCIAR UN CHOFER A UN VEHICULO
2 - AGREGAR CHOFER A LA COLA DE ESPERA
3 - VER EL SIGUIENTE CHOFER EN COLA DE ESPERA (EJEMPLO DE REJECT)
4 - CALCULAR DISTANCIA DE VIAJE
5 - INICIAR UN VIAJE
6 - FINALIZAR UN VIAJE
7 - LISTAR TELEFONOS DE LOS CLIENTES (EJEMPLO DE COLLECT)
8 - VIAJES A UNA MISMA LOCALIDAD (EJEMPLO DE SELLECT)
9 - BUSCAR CLIENTE (EJEMPLO DE DETECT)
10 - LISTAR VEHICULOS POR AÑO (EJEMPLO DE ORDENAMIENTO DE COLECCION)
11 - MOSTRAR VIAJES INICIADOS
12 - LUGARES FAVORITOS (EJEMPLO OCURRENCES OF)
13 - PARA SALIR
```

MENÚ PRINCIPAL

El inicio de un viaje implica obtener como principal medida el usuario que lo va a realizar. Del mismo se debe saber si es un usuario registrado, esto quiere decir que sus datos están almacenados en la base de datos del sistema; de ser así obtener el ID (número de identificación con el que se almacena los datos del usuario). En caso contrario se deberá cargar los datos del usuario a la base de datos del sistema para contar a posteriori con toda la información del usuario, y contar con el ID correspondiente (IMAGEN 2)

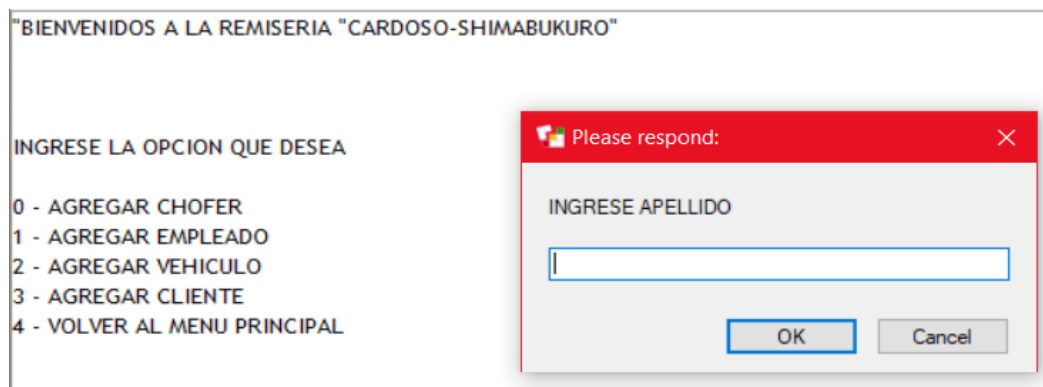


IMAGEN 2

La obtención de la ubicación de origen es un camino parecido al del usuario. En caso de que el origen sea la ubicación almacenada como domicilio, se buscará a través del ID de usuario, en cualquier otro caso, se deberá cargar en el sistema al iniciar un viaje. Lo mismo, va a suceder con la ubicación de destino (IMAGEN 3).

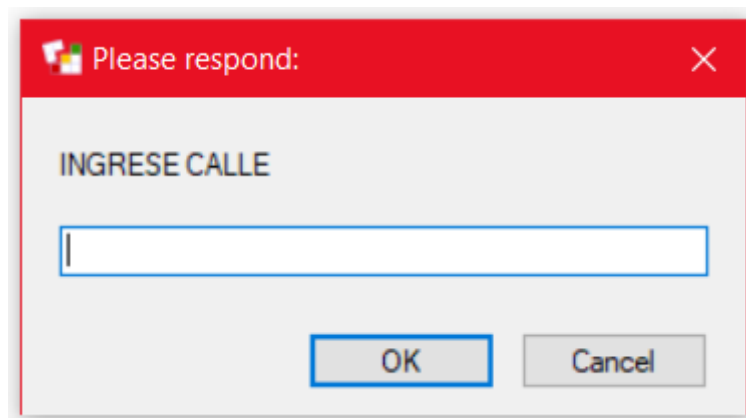


IMAGEN 3

La agencia a la hora de contratar choferes deberá cargar todos sus datos. De esta manera cada chofer va a contar con un ID identificadorio al igual que el cliente. Cabe destacar, que la información almacenada puede variar entre usuarios, choferes o empleados de la agencia (INFO CHOFER, INFO CLIENTE, INFO EMPLEADO).

self	a Chofer	self	a Cliente	self	an Empleado
getApellido	abc 'Martinez'	getApellido	abc 'Cardoso'	getAntiguedadE...	123 0
getComisionCho...	3.14 0.8	getidCliente	123 1	getApellido	abc 'Urquiza'
getHorarioChofer	abc '8 a 20'	getMail	abc 'indefinido'	getHorarioEmpl...	abc '8 a 20'
getLegajoChofer	123 1	getNombre	abc 'Mauro'	getLegajoEmple...	123 1
getMail	abc 'indefinido'	getTelefono	∞ 1558594841	getMail	abc 'indefinido'
getNombre	abc 'Lorenzo'	getUbicacion	an Ubicacion	getNombre	abc 'Roberto'
getSumaGanancias	3.14 24.533502190988	getViajesCiente	an OrderedCollection	getPuestoLabor...	abc 'telefonista'
getTelefono	∞ 1112312345			getSueldoEmple...	123 250000
getUbicacion	an Ubicacion			getTelefono	123 55555555
				getUbicacion	an Ubicacion

INFO CHOFER

INFO CLIENTE

INFO EMPLEADO

Aquí se presenta una nueva disyuntiva, ¿qué chofer deberá cubrir el viaje? En este caso, si los viajes inician desde la ubicación de la agencia, el sistema brinda la función de crear una “cola” de choferes en espera (opción 2 menú principal agrega un chofer a la cola, opción 3 muestra el siguiente), los cuales van a tener un turno para poder realizar los viajes. En el caso de que los viajes inicien desde otras ubicaciones, el sistema también brinda una función para detectar el chofer que esté desocupado (IMAGEN 4), más cerca de dicha ubicación. Así es el sistema quien se encargará, dependiendo de donde inician los viajes, a que chofer le corresponde realizarlo.

```

buscarChoferCercanoCliente: unaUbicacion
| distancia minimo chofer1 |
minimo := 99.999999.
personal keysAndValuesDo:
[:legajo :empleado1 |
empleado1 class = Chofer
ifTrue:
[distancia := (empleado1 getUbicacion) getCoordenada dist: unaUbicacion getCoordenada.
distancia < minimo
ifTrue:
[minimo := distancia.
chofer1 := empleado1]]].
^chofer1

```

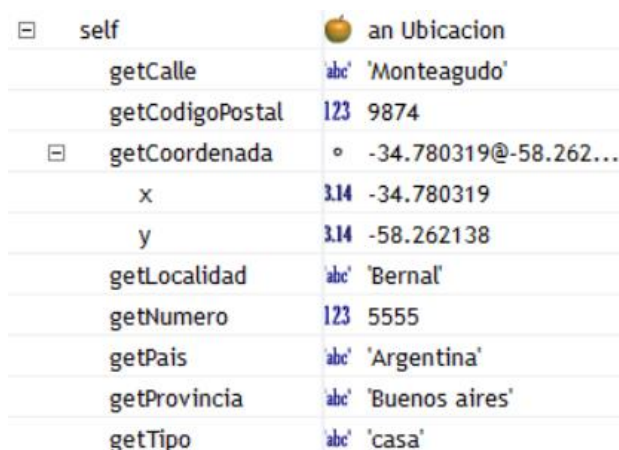
IMAGEN 4

Una vez explicado de dónde y cómo se obtienen los datos para que el sistema inicie o finalice un viaje, el menú principal es el encargado de brindar las opciones para realizarlo. La opción número 5 es la que va a iniciar el viaje. Una vez elegida, el sistema indicará que se ingrese el ID del usuario que realizará el viaje, en la pantalla se imprimirá una lista detallada de usuarios registrados para elegir entre uno de ellos. Luego, habrá que elegir el origen del viaje, si desde la agencia o desde otra ubicación (IMAGEN 5). Si elegimos el origen desde la agencia, deberemos optar si el destino es el domicilio almacenado en la información del usuario. En el caso que ambos sean afirmativos, el inicio del viaje se habrá completado satisfactoriamente.

En otros casos se deberá cargar la información requerida por el sistema, el cual una vez obtenida iniciará el viaje. Este inicio será indicado mediante un aviso.

En el caso de que el viaje haya finalizado, en la opción 6 del menú principal, se deberá optar por una opción de la lista en pantalla, el viaje iniciado al cual queremos darle un fin. También en este caso, el sistema dará un aviso.

Cuando utilizamos cualquier función del sistema, además de la funcionalidad que se elige, el sistema realiza distintas tareas. Estas tareas incluyen, en iniciar viaje el sistema también lo almacena en un diccionario con clave valor, siendo la clave el ID único del viaje y el valor los datos del viaje. O al momento de finalizarlo, calcula el importe del viaje mediante los km. recorridos y el valor del km, le agrega el viaje al cliente (una colección), como así también al historial de viajes de la agencia (un diccionario), le calcula la ganancia individual del chofer mediante el importe del viaje y la comisión particular del chofer, y lo saca de la lista de viajes iniciados.



self	an Ubicacion
getCalle	'Monteagudo'
getCodigoPostal	123 9874
getCoordenada	-34.780319@-58.262...
x	-34.780319
y	-58.262138
getLocalidad	'Bernal'
getNumero	123 5555
getPais	'Argentina'
getProvincia	'Buenos aires'
getTipo	'casa'

IMAGEN 5

Como se muestra en el menú principal, el sistema brinda distintas opciones. A pesar de que en el menú principal solo hemos abarcado algunas que consideramos más relevantes, existen muchas otras que no utilizamos para el menú. Una de estas funcionalidades que brindamos, es la “opción 0” la encargada de realizar la carga de datos para empleados, choferes, vehículos o clientes, la cual cuenta con un menú secundario (IMAGEN 6).

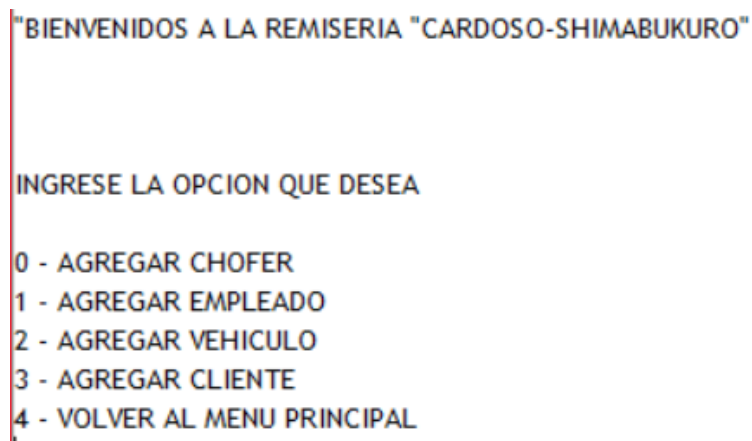


IMAGEN 6

La “opción 2” del menú principal, permite asociar a cualquier chofer con cualquier vehículo en la agencia. Esto surge de la problemática de que la agencia puede contar con vehículos propios a los cuales deberá asignarle choferes empleados.

En la “opción 4” el sistema permite a la agencia brindar presupuestos sobre viajes no iniciados, únicamente ingresando la ubicación de origen y destino. Luego el sistema informa la distancia que hay entre los dos puntos, y así permite calcular el importe al valor del Km. de la agencia.

Uno de los requisitos del “trabajo integrador” era la utilización de las funciones “Sellect, Detect, Collect y Reject”, las cuales hicimos que el sistema las permita utilizar en las opciones 7, 8, 9 y 3 (Collect, Sellect, Detect, Reject respectivamente). Con la función Collect, listamos los teléfonos de todos los clientes de la agencia, en la función Sellect permitimos ver todos los viajes realizados a una localidad ingresada por teclado, con el Detect buscamos un cliente por apellido, y el reject nos sirvió para ver el siguiente turno de la “cola” de espera.

Otro de los requisitos utilizados en el menú principal, es la utilización de una colección ordenada, la cual la podremos utilizar en la “opción 10”. Su uso nos permite ordenar los vehículos de manera ascendente o descendente, según la opción indicada, en una lista mostrada en pantalla. Y como último requisito implementado en el sistema, requisitos solicitados para el desarrollo del trabajo integrador, en la “opción 12” aplicamos la función “Occurrences of”, para listar lugares favoritos y la cantidad de veces visitados.

Este trabajo se realizó entre un equipo de trabajo. Cada entrega, cada presentación y cada función debió ordenarse y basarse en una metodología de trabajo para poder llevarse a cabo. La próxima parte del informe explica que metodología se puso en practica para solucionar los inconvenientes que pueden llegar a aparecer en un trabajo realizado en equipos. Como así también vamos a explicar el control de versiones implementado para no caer en desactualizaciones y demás inconvenientes.

METODOLOGÍAS DE TRABAJO

Durante el transcurso del proyecto realizamos iteraciones cortas de una semana, durante las mismas la mayor parte la dedicamos al desarrollo y el último día a integrar lo que hicimos y a probar que todo compile y funcione sin errores. Algo a mejorar fue la planificación ya que empezamos a escribir código antes de especificar y organizar las tareas. Cada uno fue trabajando sobre los incrementos que iba haciendo el otro y creando e implementando tareas que creíamos que necesitaba el proyecto.

Podemos decir que tomamos los sprint, aunque más cortos, de scrum tratando de entregar un producto funcional todas las semanas. Por otro lado, asumimos los roles de cliente, programador y tester, tratando de mantener un ritmo constante como establece XP (programación extrema). Sin embargo, no especificamos las historias de usuario de XP, ni un “product backlog” o “sprint backlog” de Scrum, sino que nos guiamos en un comienzo por el diagrama de clases para escribir el código de la aplicación.

Por último, las presentaciones semanales en clase serían como la demostración de requisitos completados de la metodología scrum.

Una de las herramientas que utilizamos con mas frecuencia y a simple vista, fue la integración continua. Consiste en hacer la compilación y ejecución de pruebas automáticamente lo más a menudo posible para detectar errores cuanto antes. Implica la fusión varias veces al día de las ramas. Descargar el código cada x tiempo, compilarlo, ejecutar las pruebas y realizar informes. De esta manera cada integrante del equipo se aseguraba de contar con la última versión del código. Esta última versión sin errores, porque ya paso las pruebas correspondientes. Cada versión que se subía era probada y cada integrante realizaba los informes correspondientes. Entonces, mediante la integración continua pudimos asegurarnos de detectar y solucionar los problemas de forma inmediata, justo antes de compartir el avance, contando siempre con una versión prototípica sin errores.

FRAMEWORKS

Frameworks que se podrían utilizar para el desarrollo orientado a objetos:

- Dolphin Smalltalk

Dolphin es un framework para el lenguaje orientado a objetos Smalltalk en Windows. Es un IDE (Integrated Development environment) que incluye una máquina virtual con una aplicación que muestra una GUI (interfaz gráfica de usuario) con diferentes herramientas como: un explorador de clases, un buscador, un monitor de procesos, un system transcript, un workspace y muchas otras herramientas que nos permiten escribir, ejecutar, testear y debuggear el código escrito en Smalltalk.

Otro framework destinado a Smalltalk muy parecido a Dolphin es Pharo.

- Hibernate

Es un framework de mapeo objeto-relacional (ORM) para Java. El mapeo objeto-relacional es una técnica de programación que permite convertir datos entre modelos diferentes que conviven en una app: la programación orientada a objetos y la base de datos relacional. En este caso, transforma los datos entre los tipos utilizados por Java y por SQL.

También existe una versión NHibernate para trabajar con el lenguaje C# para su integración en la plataforma .NET.

- Qt

Es un framework de desarrollo de software para C++, contiene un conjunto entendible, intuitivo y modularizado de librerías de clases para C++, y está llena de APIs (application programming interface) que simplifica el desarrollo de software. Con Qt es posible producir un código muy legible, reusable y mantenible con buen rendimiento y multiplataforma.

Dentro de este framework podemos encontrar entre otras cosas:

- Herramientas de diseño para crear UIs y/o GUIs;
- Herramientas para el desarrollo: compiladores, debuggers, herramientas de testing automático, conversores de formato qml, un ide, un traductor, etc.

USO DE PATRONES

Para el desarrollo del problema planteado, hemos identificado la posible utilización de algunos patrones de diseño que simplificarán la implementación.

Strategy

Uno de estos posibles patrones que se podrían implementar y simplificar la tarea, es el patrón “Strategy”. Con la implementación de éste patrón, podríamos crear un solo método, “agregar” por ejemplo, y pasar una opción por parámetro para que, según la opción, agreguemos un objeto de una clase u otra. Podríamos agregar un “chofer”, un “empleado”, o un “cliente”. Cada uno de ellos deberá tener una implementación particular, pero a través de un solo método. En la clase “remisería” creamos los métodos “agregar, eliminar, está vacía, asignar, buscar, existe, cargar Datos”. Estos métodos, para cada clase se implementan de manera distinta, pero con el uso del patrón mencionado, se podría hacer genérico y que cada clase lo implemente de manera individual.

Factory method

Otro patrón muy utilizado es el “Factory Method”. Factory method es un patrón de creación que define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase de objetos instanciar.

Dentro de nuestro sistema podríamos utilizar este patrón para la creación de instancias de Chofer, Empleado o Cliente que son subclases de Persona. Deberíamos crear una clase abstracta FabricaDePersonas y una fábrica para Chofer, otra para Cliente y una última para Empleado. El código de la FabricaDePersonas se vería algo así:

```
Abstract class FabricaDePersonas{  
    Static Persona crearPersona(int quePersona){  
        FabricaDePersona fabrica = null;  
        if(quePersona == 'Chofer')  
            fabrica = new FabricaDeChofer();
```

```

        if(quePersona == 'Cliente')

            fabrica = new FabricaDeCliente();

        if(quePersona == 'Empleado')

            fabrica = new FabricaDeEmpleado();

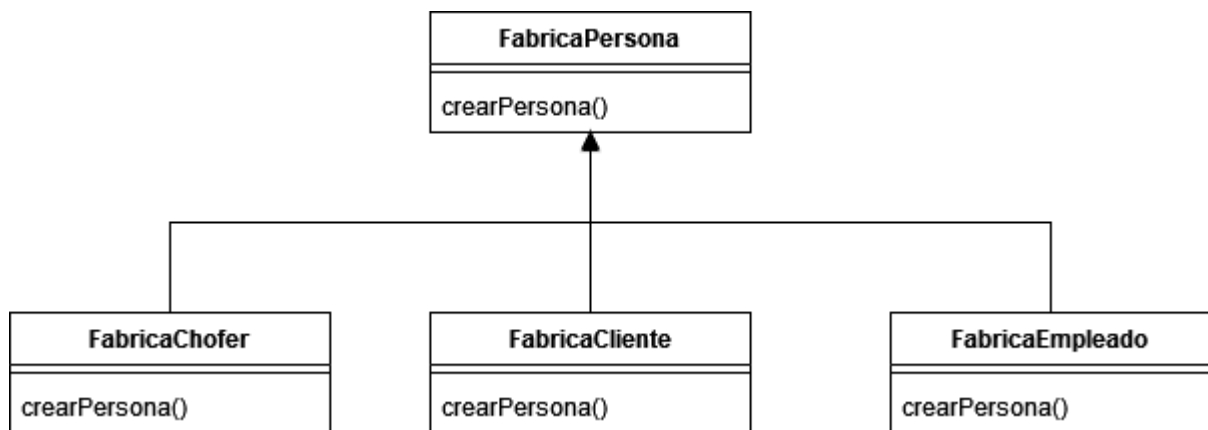
        return fabrica.crearPersona()

    }

    Abstract Persona crearPersona()

}

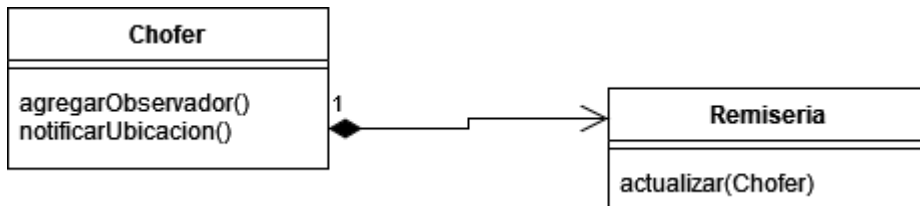
```



Observer

El patrón Observer es un patrón de comportamiento que define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y se actualicen automáticamente todos los objetos que dependan de él.

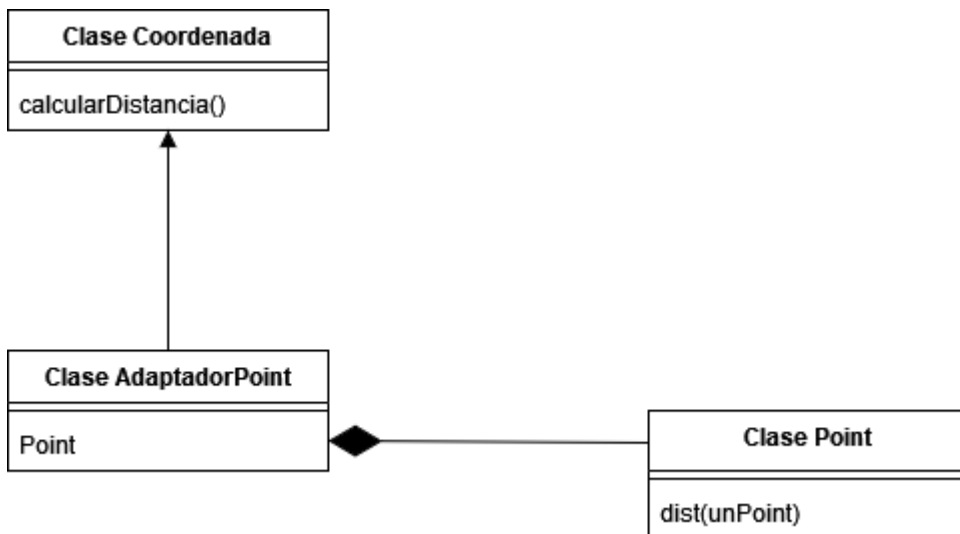
En nuestro sistema podríamos usar este patrón para notificar a la remisería de todos los cambios de estado de ubicación (coordenadas GPS) de los choferes.



Adapter

El patrón adapter es un patrón estructural que convierte la interfaz de una clase en otra interfaz que es la que esperan los clientes. Permite que cooperen clases que de otra forma no podrían por tener interfaces incompatibles.

Pudimos haber usado este patrón para trabajar dentro de la clase Ubicación, por ejemplo, podríamos crear una clase Coordenada que adapte la clase Point, específicamente usamos el método “dist: aPoint” de la clase Point para calcular la distancia entre la ubicación origen y la ubicación destino del cliente.



Singleton

Es un patrón creacional que garantiza que una clase tenga una única instancia y proporciona un único punto de acceso a ella.

Podemos usar Singleton para asegurarnos de que sólo creemos una única instancia de Remisería. Esta clase solo es necesario crearla una vez, para así evitar duplicaciones y errores.

El código sería algo así:

Clase Remisería

```
Remiseria_insta := nil;
```

```
GetInstancia
```

```
(Remiseria_insta == nil) ifTrue:
```

```
    ^self new crearRemiseriaValorKM: unValorKM nombreRemiseria:  
    unNombre
```

```
ifFalse:
```

```
    ^Remisería_insta;
```

Iterator

Por último, el patrón “Iterator” siempre es una muy buena herramienta a la hora de recorrer colecciones. En el sistema se necesitó recorrer diccionarios de distintas maneras y en muchos métodos. Con este patrón, podríamos haber tenido una solución muy eficaz.

¿QUÉ REPOSITORIOS SE PODRÍAN UTILIZAR?

Un servicio de alojamiento de repositorios es una herramienta de gestión organizativa que ofrece una visión transparente en el proceso de flujo de trabajo del desarrollo de software mejorando el sistema de control de versiones. Un servicio de alojamiento de repositorios tiene herramientas para medir, monitorizar, debatir y gestionar la eficiencia y precisión del desarrollo del software.

Como se señaló más arriba cada servicio de alojamiento de repositorio usa un sistema de control de versiones. Un **sistema de control de versiones (VCS)** es una herramienta de software que monitoriza y gestiona cambios en un sistema de archivos. Un VCS ofrece herramientas para compartir e integrar cambios en otros usuarios del VCS. Un repositorio es un sistema de archivo que está siendo monitorizado por el VCS. Un VCS monitorizará las adiciones, eliminaciones y modificaciones de las líneas de texto que contiene cualquier archivo de código fuente de nuestra aplicación. El VCS más conocido es Git.

Al elegir un servicio de alojamiento de repositorio de código tendremos que tener en cuenta los siguientes puntos:

- Sistema de control de versiones compatible
- Tamaño del equipo y control de acceso
- Programación de publicación
- Tamaño del proyecto y almacenamiento de datos
- Herramientas externas e integraciones de terceros

En particular, vamos a tratar tres soluciones de repositorios de código: Bitbucket, GitHub y GitLab. Todas nos permiten alojar proyectos utilizando el *sistema de control de versiones* Git.

BITBUCKET VS GITHUB VS GITLAB

- **Niveles de autenticación:** GitHub te da la opción de otorgar acceso según se quiera leer o escribir, con GitLab puedes proporcionar acceso asignando diferentes roles.
- GitLab permite incluir adjuntos en un *issue* o problema.
- GitLab permite marcar un proyecto como que está en desarrollo, lo que avisará a otros usuarios de la situación del mismo. Esto permite evitar que otros usuarios integren tu proyecto antes de tiempo...
- GitHub tiene una comunidad de usuarios y una cantidad de proyectos muy superior a GitLab. Mientras que en GitLab hay poco más de 100.000 proyectos, en GitHub superan los 35.000.000.
- Bitbucket se integra fácilmente a Jira, la herramienta para desarrollo ágil más conocida.
- Bitbucket garantiza que no se expondrá tu código con una auditoría de terceros de SOC 2 Tipo II (controles de sistemas y organizaciones) es un informe que se centra en los controles de creación de informes relacionados con la seguridad, disponibilidad y confidencialidad.

Cuadro comparativo

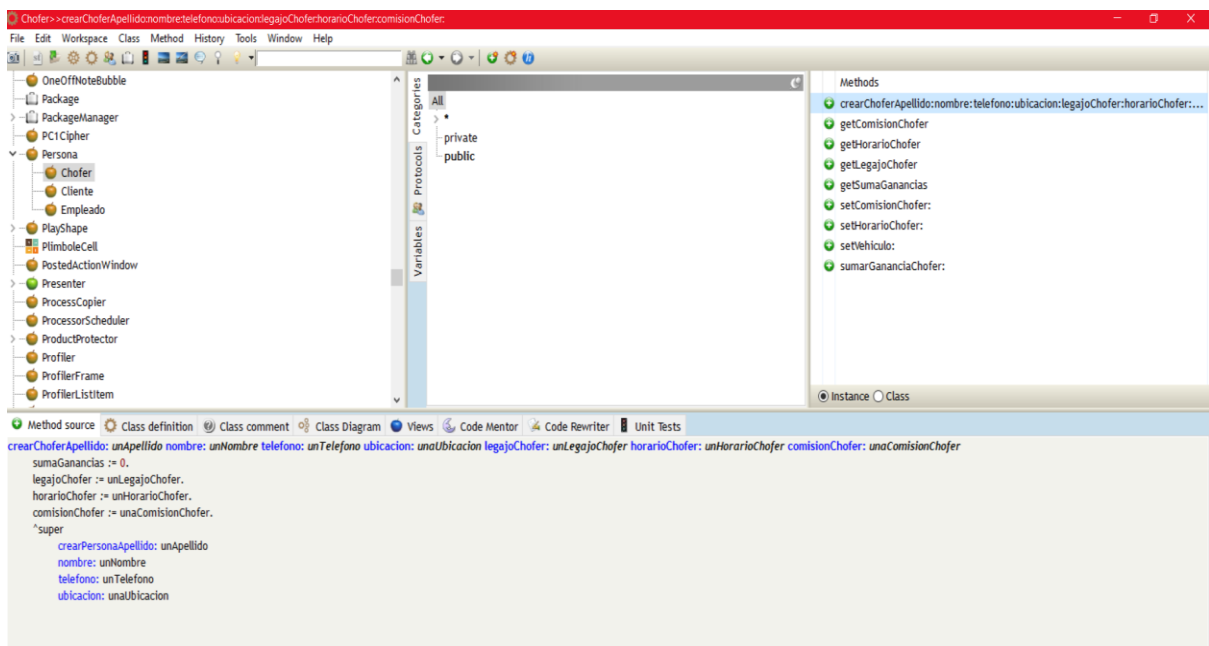
CARACTERISTICAS	BITBUCKET	GITLAB	GITHUB
Open Source	No es open source	Gratis y open source	No es open source
Integración continua y entrega (CI/CD)	SI	SI	NO
Autenticación y control de accesos	SI	Acceso acorde a los roles	Acceso según lectura o escritura
Revisión de código	SI	SI	SI
Fácil documentar	SI	SI	SI
Integración con Jira	SI	NO	NO
Repositorios privados gratuitos	SI	SI	SI
Certificación Soc 2 Tipo II	SI	NO	NO
Alojamiento propio	Bitbucket server	Solo para empresas	SI
Comunidad	10 millones de usuarios	500 mil usuarios	40 millones de usuarios
Rendimiento	Lento	Lento	Rápido
Precio	Buena relación costo-rendimiento	Económico	Más caro
Tutoriales y guías	No las suficientes	Pocas	Mucha información
Interfaz de usuario	Compleja	Fácil de usar	Fácil de usar

ANEXOS

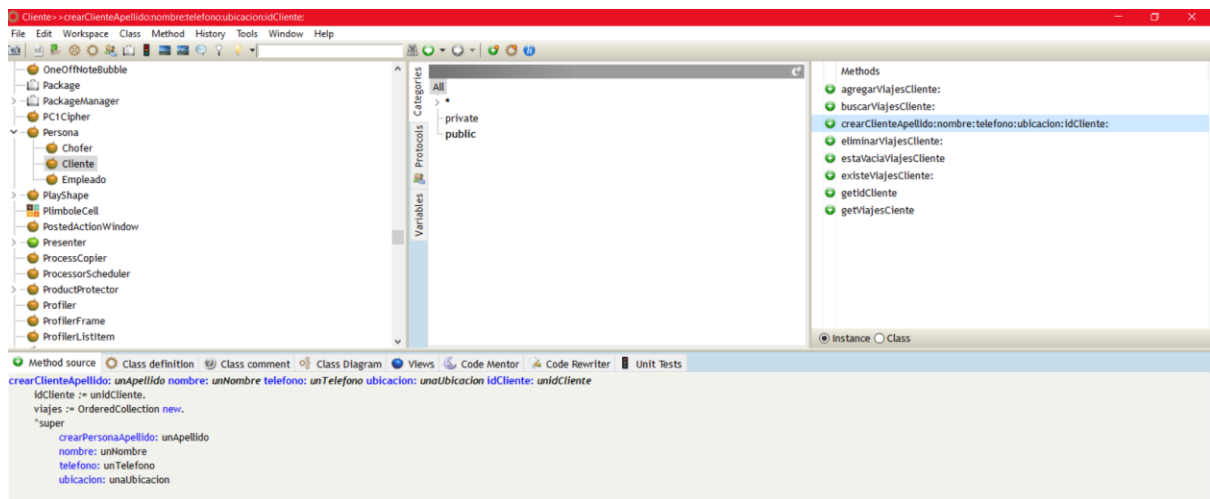
ANEXO 1: DIAGRAMA REMISERIA.DRAIO (adjunto)

ANEXO 2: DIAGRAMA SECUENCIA REMISERIA.DRAIO (adjunto)

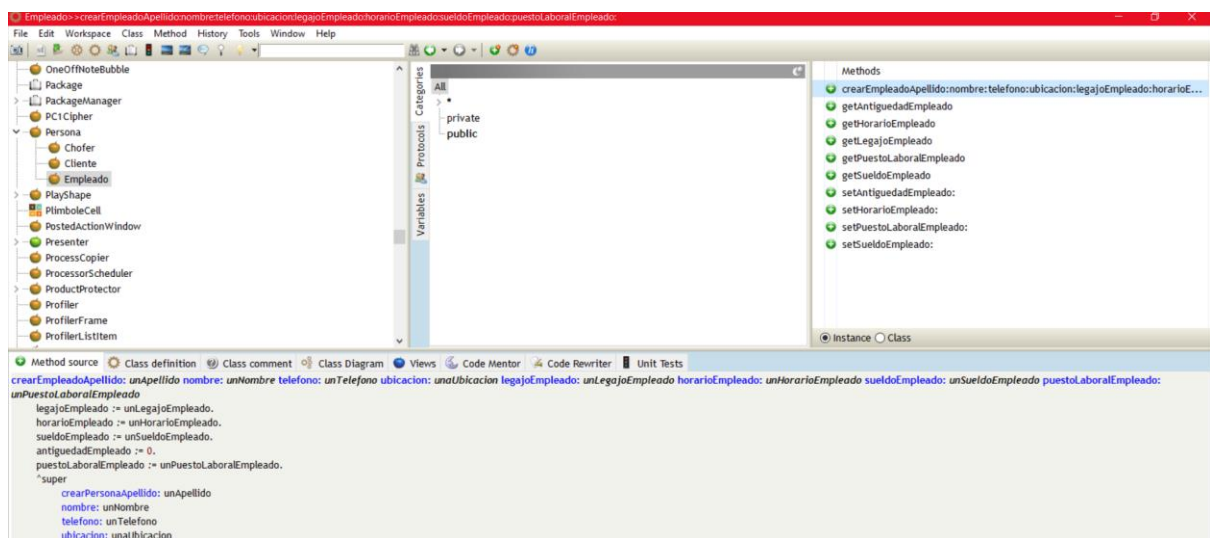
ANEXO 3:



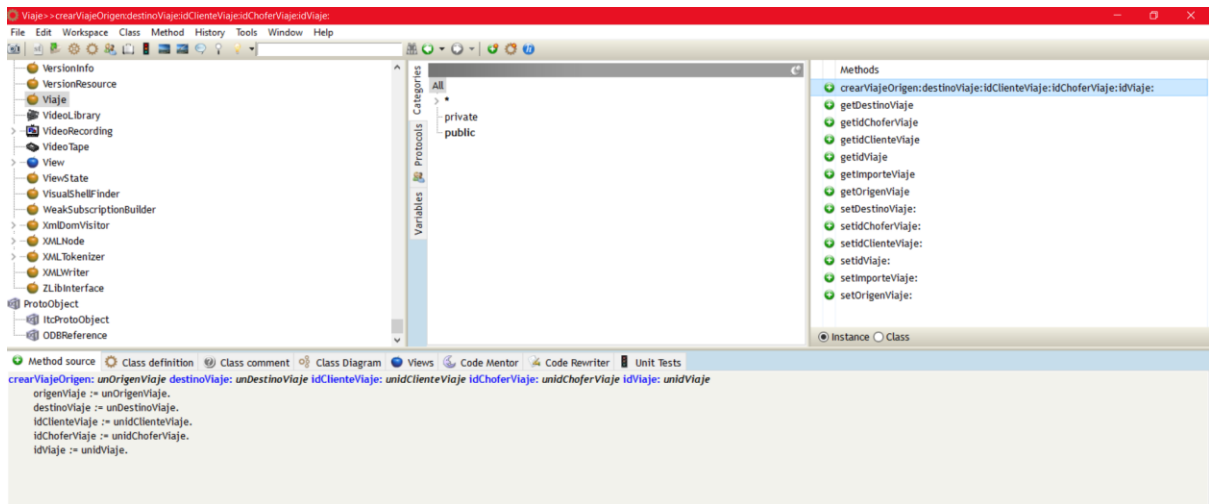
SUBCLASE CHOFER



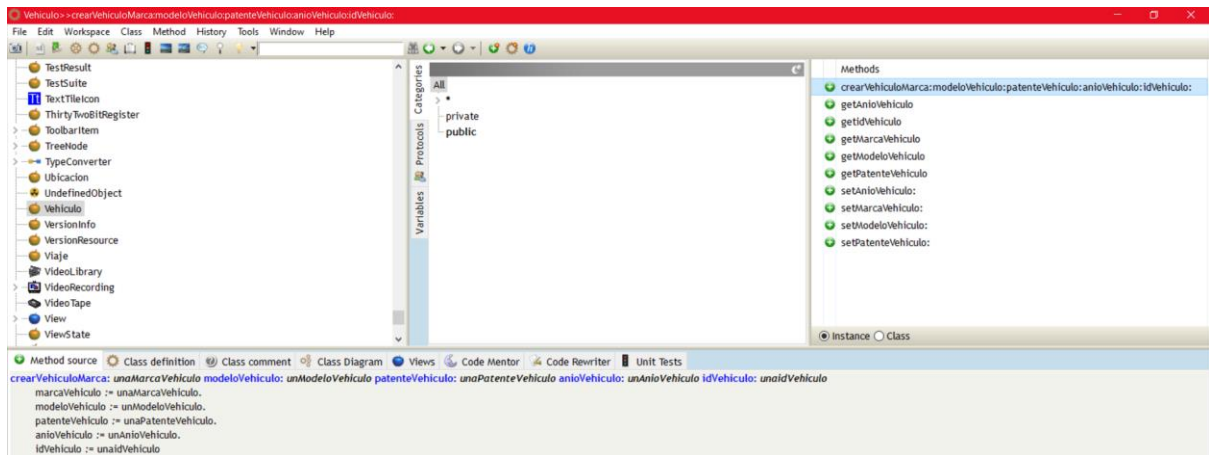
SUBCLASE CLIENTE



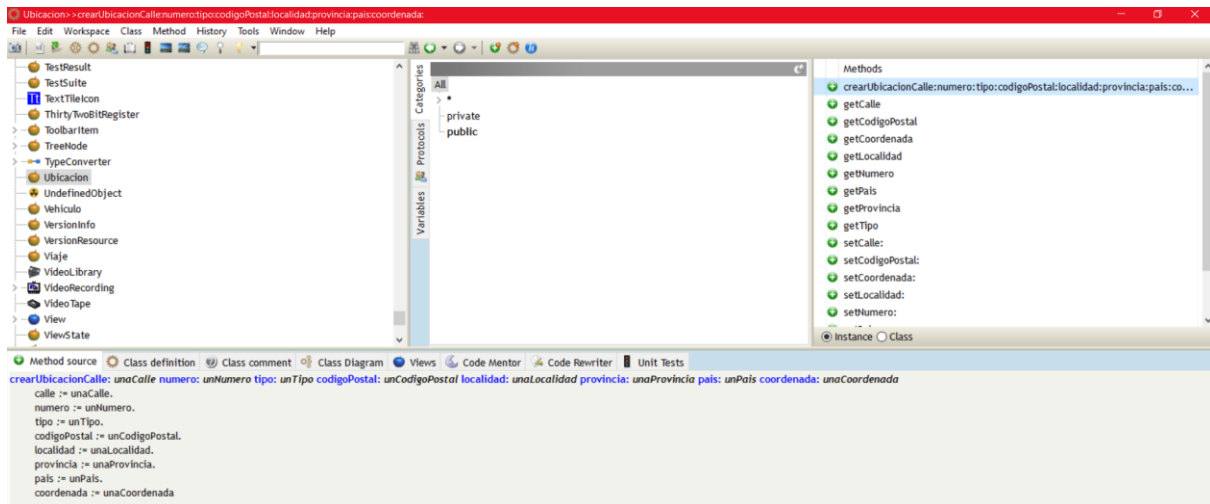
SUBCLASE EMPLEADO



CLASE VIAJE



CLASE VEHICULO



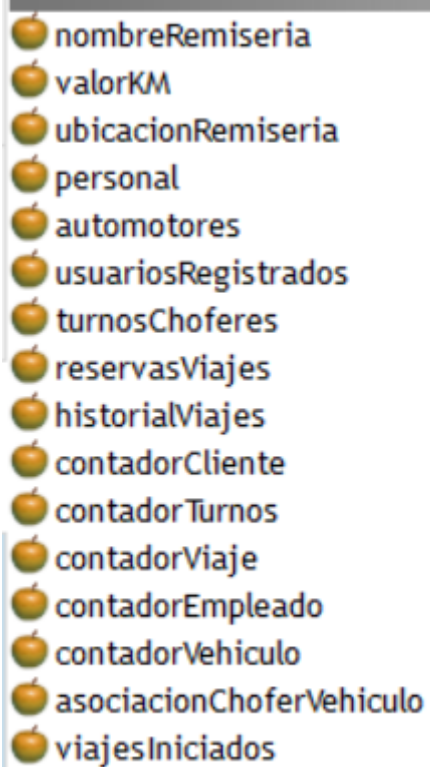
CLASE UBICACIÓN

```

crearRemiseriaValorKM: unValorKM nombreRemiseria: unNombre
    contadorCliente := 1.
    contadorTurnos := 1.
    contadorViaje := 1.
    contadorEmpleado := 1.
    contadorVehiculo := 1.
    personal := Dictionary new.
    automotores := Dictionary new.
    usuariosRegistrados := Dictionary new.
    turnosChoferes := OrderedCollection new.
    reservasViajes := OrderedCollection new.
    historialViajes := Dictionary new.
    valorKM := unValorKM.
    nombreRemiseria := unNombre.
    ubicacionRemiseria := Ubicacion
        constructorUbicacionCalle: 'calchaqui'
        numero: 6200
        tipo: 'universidad'
        codigoPostal: 'indefinido'
        localidad: 'Florencia Varela'
        provincia: 'Buenos Aires'
        pais: 'Argentina'
        coordenada: -34.774320 @ -58.267148.
    asociacionChoferVehiculo := Dictionary new.
    viajesIniciados := OrderedCollection new.

```

CONSTRUCTOR DE INSTANCIA CLASE REMISERIA



- nombreRemiseria
- valorKM
- ubicacionRemiseria
- personal
- automotores
- usuariosRegistrados
- turnosChofers
- reservasViajes
- historialViajes
- contadorCliente
- contadorTurnos
- contadorViaje
- contadorEmpleado
- contadorVehiculo
- asociacionChoferVehiculo
- viajesIniciados

VARIABLES DE INSTANCIA CLASE REMISERIA

agregarAutomotorRemiseria:	cargarDatosIniciarViajeRemiseria	imprimirViajesIniciados
agregarHistorialViajeRemiseria:	cargarDatosViajesAunaMismaLocalidad	iniciarUnViajeRemiseriaOrigen: Remiseria>
agregarPersonalRemiseria:	cargarDatosYcrearUbicacionRemiseria	listarTelefonosClientes
agregarUsuarioRemiseria:	crearRemiseriaValorKM:nombreRemiseria:	listarVehiculos:
agregarViajesIniciadosRemiseria:	eliminarAutomotorRemiseria:	lugaresFavoritos
asignaridClienteRemiseria	eliminarPersonalRemiseria:	menuCargaDatos
asignaridVehiculoRemiseria	eliminarTurnoChoferes	menuPrincipalRemiseria
asignaridViajeRemiseria	eliminarUsuarioRemiseria:	modificarAutomotorRemiseria:
asignarLegajoEmpleadoRemiseria	eliminarViajesIniciadosRemiseria:	modificarPersonalRemiseria:
asignarTurnoRemiseria:	existeAutomotorRemiseria:	modificarUsuariosRegistradosRemiseria:
asociarChoferVehiculo:vehiculo:	existePersonalRemiseria:	modificarViajesIniciadosRemiseria:
automotoresEstaVaciaRemiseria	existeUsuarioRemiseria:	mostrarViajesIniciados
buscarAutomotoresRemiseria:	existeViajesIniciadosRemiseria:	personalEstaVaciaRemiseria
buscarChoferCercanoCliente:	finalizarViajeRemiseria:	setNombreRemiseria:
buscarPersonalRemiseria:	getAsociacionChoferVehiculo	setUbicacionRemiseria:
buscarReservaPorCliente:	getAutomotoresRemiseria	setValorKMRemiseria:
buscarUsuarioRemiseria:	getContadorClienteRemiseria	turnosChoferesEstaVacia
buscarViajesIniciadosRemiseria:	getHistorialViajes	usuariosRegistradosEstaVaciaRemiseria
calcularDistanciaOrigen:destino:	getNombreRemiseria	verSiguienteTurnoRemiseria
calcularImporteViaje:	getPersonalRemiseria	viajesAunaLocalidad:
cargaAutomatica	getTurnosChoferes	viajesIniciadosEstaVaciaRemiseria
cargaDatosAgregarPersonal:	getUbicacionRemiseria	
cargarDatosAgregarAutomotor	getUsuariosRegistrados	
cargarDatosAgregarUsuario	getValorKMRemiseria	
cargarDatosAsignarTurnoRemiseria	getViajesIniciados	
cargarDatosAsociarChoferVehiculoRemiseria	historialViajesEstaVacia	
cargarDatosBuscarCliente	imprimirChoferes	
cargarDatosCalcularDistanciaOrigen	imprimirChoferesVehiculos	
cargarDatosFinalizarViajeRemiseria	imprimirUsuarios	

METODOS DE CLASE REMISERIA

[VOLVER AL TEXTO](#)

ANEXO 4:

```

verSiguienteTurnoRemiseria
  | valor |
  valor := turnosChoferes last.
  turnosChoferes := turnosChoferes reject: [:int | int = valor].
  Transcript clear.
  Transcript
    show: "*****".
  Transcript cr.
  Transcript
    show: 'EL SIGUIENTE CHOFER ES ', valor getApellido printString , ' ', valor getNombre printString.
  Transcript cr.
  Transcript
    show: "*****".
  ^valor

```

EJEMPLO DE REJECT CLASE REMISERIA

listarTelefonosClientes

```
| coleccion |
self imprimirChoferes.
coleccion := usuariosRegistrados collect:
    [:cliente |
        cliente getApellido printString , ' ' , cliente getNombre printString , ' ' , cliente getTelefono printString].

Transcript clear.
Transcript show: 'APELLIDO Y NOMBRE' , ' ' , ' ' , 'TELEFONO'.
Transcript cr.
Transcript show: '-----'.
Transcript cr.
coleccion do:
    [:telefono |
        Transcript show: telefono printString.
        Transcript cr.
        Transcript show: '*****'.
        Transcript cr].
^coleccion
```

EJEMPLO DE COLLECT CLASE REMISERIA

cargarDatosViajesAunaMismaLocalidad

```
| aux opcion |
opcion := Prompter prompt: 'INGRESE LA LOCALIDAD DESEADA'.
aux := historialViajes select: [:viajes | viajes getDestinoViaje getLocalidad = opcion].
Transcript clear.
Transcript show: 'ID VIAJE ' , ' ' | LOCALIDAD'.
Transcript cr.
Transcript show: '-----'.
Transcript cr.
aux do:
    [:each |
        Transcript show: ' ' , each getidViaje printString , ' ' , each getDestinoViaje getLocalidad printString.
        Transcript cr.
        Transcript show: '*****'.
        Transcript cr].
^aux
```

EJEMPLO DE SELECT CLASE REMISERIA

cargarDatosBuscarCliente

```
| aux idUnico |
self imprimirUsuarios.
idUnico := Prompter prompt: 'INGRESE EL ID DEL CLIENTE QUE DESEA BUSCAR'.
aux := usuariosRegistrados detect: [:cliente | cliente getIdCliente = idUnico asNumber].
"*****"

Transcript clear.
Transcript show: 'APELLIDO ', ' ' | NOMBRE ', ' | TELEFONO ', ' | E-MAIL '.
Transcript cr.
Transcript show: '-----'.
Transcript cr.
Transcript
    show: ' ', aux getApellido printString, ' ', aux getNombre printString, ' ',
        , aux getTelefono printString, ' ',
        , aux getMail printString.

Transcript cr.
Transcript show: "*****".
Transcript cr.
"*****"

^aux
```

EJEMPLO DE DETECT CLASE REMISERIA

```

listarVehiculos: unaOpcion
    "Tomo los vehiculos y los ordeno por año"

    | orCol_autoAnio sortCol_autoAnio str str2 |
    orCol_autoAnio := OrderedCollection new.
    self getAutomotoresRemiseria keysAndValuesDo: [:eachKey :eachValue | orCol_autoAnio add: eachValue].
    "orCol_autoAnio inspect."
    sortCol_autoAnio := SortedCollection new.
    unaOpcion = '0'
        ifTrue:
            [sortCol_autoAnio := orCol_autoAnio
                asSortedCollection: [:a :b | a getAnioVehiculo < b getAnioVehiculo]].
    unaOpcion = '1'
        ifTrue:
            [sortCol_autoAnio := orCol_autoAnio
                asSortedCollection: [:a :b | a getAnioVehiculo > b getAnioVehiculo]].
    "sortCol_autoAnio inspect."
    Transcript clear.
    Transcript show: 'MODELO ', '          | PATENTE ', '          | AÑO'.
    Transcript cr.
    Transcript show: '-----'.
    Transcript cr.
    sortCol_autoAnio do:
        [:each |
            str := each getMarcaVehiculo , ' ', each getModeloVehiculo , '          | ', each getPatenteVehiculo
                , '          | '.
            str2 := each getAnioVehiculo printString.
            Transcript show: str , '.
            Transcript show: str2.
            Transcript cr.
            Transcript show: '-----'.
            Transcript cr]

```

EJEMPLO SORTED COLLECTION CLASE REMISERIA

```

lugaresFavoritos
    | lugaresPreferidos setLugaresPreferidos diccionarioLugaresPreferidos |
    lugaresPreferidos := OrderedCollection new.
    historialViajes
        keysAndValuesDo: [:id :viajes | lugaresPreferidos add: viajes getDestinoViaje getLocalidad].
    setLugaresPreferidos := lugaresPreferidos asSet.
    diccionarioLugaresPreferidos := Dictionary new.
    setLugaresPreferidos
        do: [:lugar | diccionarioLugaresPreferidos at: lugar put: (lugaresPreferidos occurrencesOf: lugar)].
    Transcript clear.
    Transcript show: 'LUGAR VISITADO ', '          | CANTIDAD DE VECES VISITADO'.
    Transcript cr.
    Transcript show: '-----'.
    Transcript cr.
    diccionarioLugaresPreferidos keysAndValuesDo:
        [:id :cantidad |
            Transcript show: ' ', id , '          ', cantidad printString.
            Transcript cr.
            Transcript show: '*****'.
            Transcript cr]

```

EJEMPLO DE OCCURRENCES OF CLASE REMISERIA

[VOLVER AL TEXTO](#)