

# Proyecto Plataforma para Taller Automotor

**Materia: Diseño Orientado a Objetos.**

**Integrantes: Álvarez Tomás, Savino Bautista, Rocha Martín, Szulman Pablo.**

CONTENIDO

[Proyecto Plataforma para Taller Automotor 1](#_Toc75358663)

[**INTRODUCCIÓN 2**](#_Toc75358664)

[DIAGRAMA DE CLASES 2](#_Toc75358665)

[Grafico de Diagrama de Clases](#_Toc75358666) 3

[**DIAGRAMAS DE SECUENCIA 4**](#_Toc75358667)

[DIAGRAMA DE SECUENCIA: ENTREGA VEHÍCULO 4](#_Toc75358668)

[DIAGRAMAS DE SECUENCIA: REGISTRAR TURNO 5](#_Toc75358669)

[**DIAGRAMA DE ESTADO** 5](#_Toc75358670)

[DIAGRAMA DE TURNO 5](#_Toc75358672)

[**PATRONES DE DISEÑO (Factory Method ) 6**](#_Toc75358673)

[**DAO (Data access object)** 7](#_Toc75358675)

[**MODELO-VISTA-CONTROLADOR (MVC)** **8**](#_Toc75358676)

[**CASO DE USO** **9**](#_Toc75358677)

[**DIAGRAMA BASE DE DATOS** **10**](#_Toc75358677)

[**TEST** **11**](#_Toc75358677)

## 

## INTRODUCCIÓN

Hemos desarrollado un proyecto integrador para un Taller Automotor cumpliendo los requisitos solicitados. Para comenzar a abordar y tener una mejor visión del proyecto explicaremos los elementos y herramientas usadas.

Las personas deben tener la posibilidad de sacar turnos para control o arreglos de su automotor, que contará con una póliza de seguro, el taller debe brindarle información de disponibilidad, registrar entada y salida del mismo, una constancia de conformidad, y deberá contar con información de las compañías de seguro para los convenios, así también como de sus mecánicos, clientes y empleados.

Para comenzar la explicación detallada, mostraremos el diagrama de clases usadas y sus relaciones.

## DIAGRAMA DE CLASES

El diagrama de clases recoge las clases de objetos y sus asociaciones. En este diagrama se representa la estructura y el comportamiento de cada uno de los objetos del sistema y sus relaciones con los demás objetos, pero no muestra información temporal.

Especificaciones: Una clase abstracta es una clase que no tiene instancias. Su utilidad consiste en proveer estructura y comportamiento común a todas las subclases que heredan de ella. Por éste motivo hemos creado la clase abstracta Persona y Empleado.

Cada taller tendrá sus respectivos clientes, los cuales tendrán un vehículo con póliza, dicha póliza pertenece a alguna compañía de seguros.

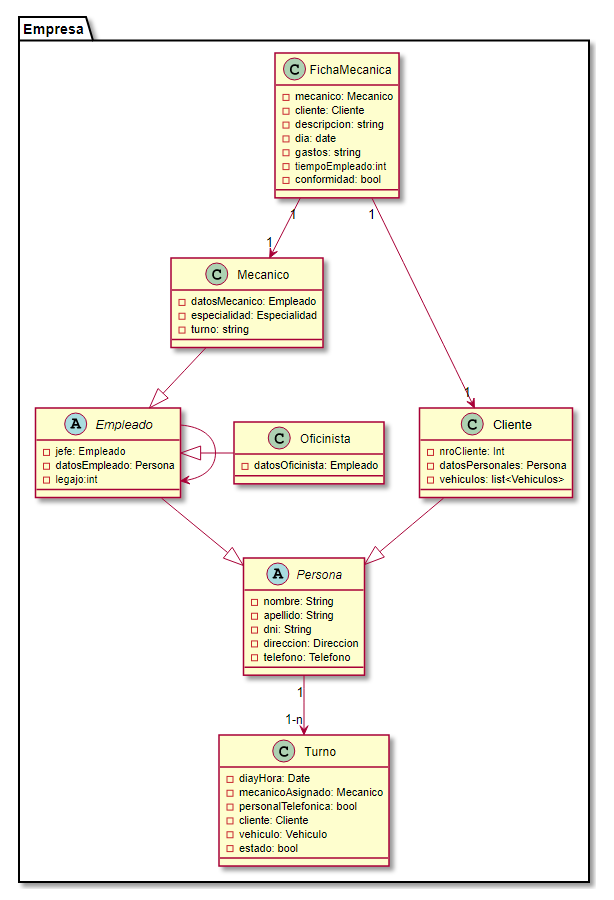
Cada persona puede tener uno o varios turnos registrados en la agenda, tanto los empleados como los clientes. Cada persona tendrá sus datos personales.

Contamos con una ficha técnica entregada al mecánico y al cliente al finalizar el trabajo. Cada mecánico tendrá su especialidad.

En el taller diferenciaremos a los empleados por aéreas (mecánico, encargado, oficinista, etc). El taller contará con la lista de las compañías de seguro que posean convenios.

## 

## DIAGRAMA DE CLASES



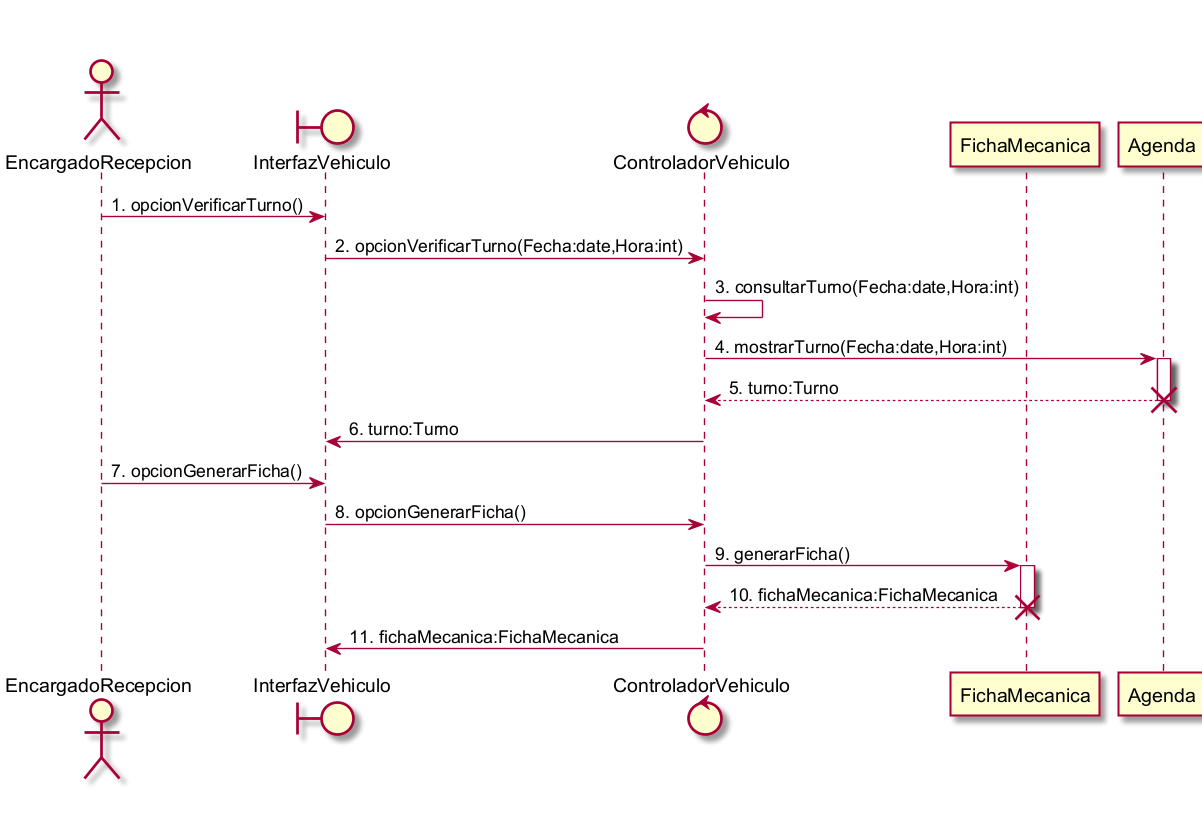
## DIAGRAMAS DE SECUENCIA:

Breve explicación: El diagrama de secuencia es un tipo de diagrama de interacción cuyo objetivo es describir el comportamiento dinámico del sistema de información haciendo énfasis en la secuencia de los mensajes intercambiados por los objetos.

## DIAGRAMA DE SECUENCIA: ENTREGA VEHÍCULO

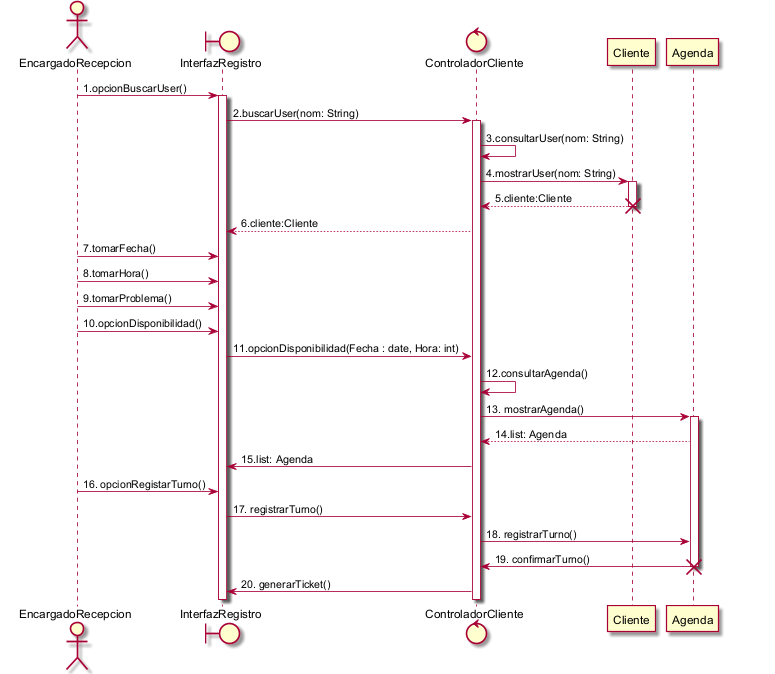
El encargado de la recepción podrá registrar una entrega de vehículo al cliente y brindarle una ficha de conformidad con los detalles del trabajo, el mecánico que lo realizó, turno solicitado, precio, entre otros, por lo cual procederemos a explicarlo con el siguiente diagrama de secuencia:

El encargado deberá verificar que el trabajo fue realizado para un turno ya anotado en la Agenda, el mismo deberá ser mostrado ya que la Agenda contamos con todos los datos del cliente y del automotor; una vez consultado procederá a generar una ficha mecánica la cual contará con todos los datos para posteriormente entregarle la misma al cliente en modo de factura o control, y así registrar y tener orden de las operaciones realizadas en el taller.



## DIAGRAMAS DE SECUENCIA: REGISTRAR TURNO

El encargado de la recepción deberá registrar una turnos de las personas que quieran realizar cualquier tipo de control o arreglo en su vehículo, deberá constatar disponibilidad, tomar datos, entre otras cosas por lo cual procederemos a explicarlo con el siguiente diagrama de secuencia:



El encargado deberá verificar si la persona solicitante está registrada como cliente, por lo cual deberá tener control de los clientes ya registrados, se deberá tomar la fecha que solicita la persona y el problema de su vehículo, el encargado tendrá control de la agenda para verificar disponibilidad del turno solicitado, en caso de encontrarla se asignará el turno y si no se buscará otra fecha/hora, una vez confirmado el turno se generará un ticket con los datos del turno al cual tendrá acceso el encargado para brindárselo al cliente.

## DIAGRAMA DE ESTADO

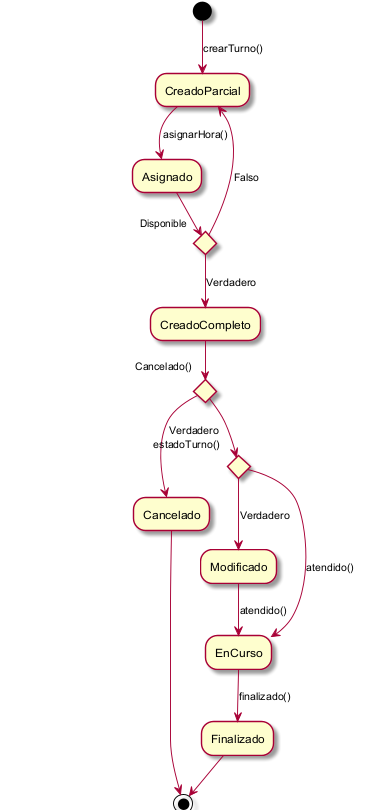
Se usan para representar estados de objetos.

## DIAGRAMA DE TURNO

El turno será creado y pasará a Creado Parcial, una vez que se le asigna la hora pasa a Asignado donde consulta disponibilidad, si no la hay volverá a Creado Parcial, si hay disponibilidad pasará a Creado Completo.

El turno ya ha sido creado en su totalidad.

Luego si el turno es Cancelado pará a estado cancelado; si no ha sido cancelado puede haber sido modificado, sea modificado o no, siempre y cuando mientras no sea cancelado, seguirá su curso, estará en estado En Curso hasta ser Finalizado.



## PATRONES DE DISEÑO

Hemos implementado Factory Method a la clase Persona.

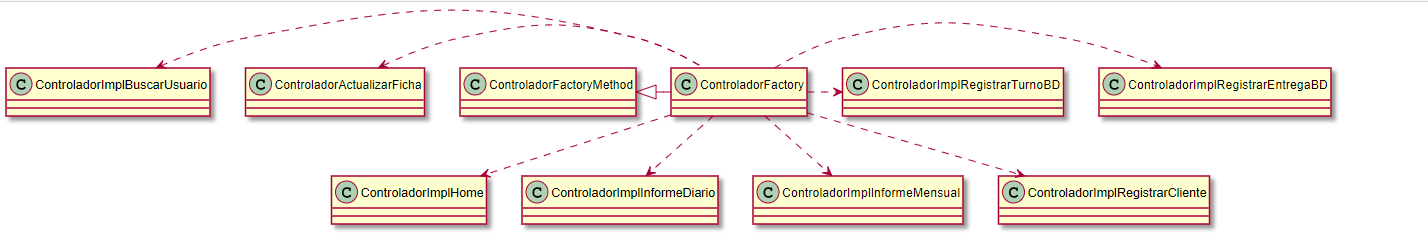
Factory Method es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

Para persona, lo usamos para “delegar” a subclases la creación de distintos tipos de Persona dependiendo del rol que cumpla al interactuar.

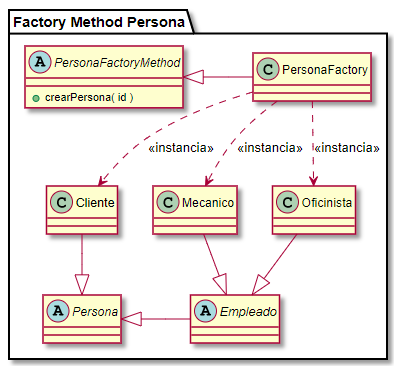
**Factory Method se usa:**

* Cuando una clase no puede adelantar las clases de objetos que debe crear.
* Cuando una clase pretende que sus subclases especifiquen los objetos que ella crea.
* Cuando una clase delega su responsabilidad hacia una de entre varias subclases auxiliares y queremos tener localizada a la subclase delegada.

## Factory Method Controlador



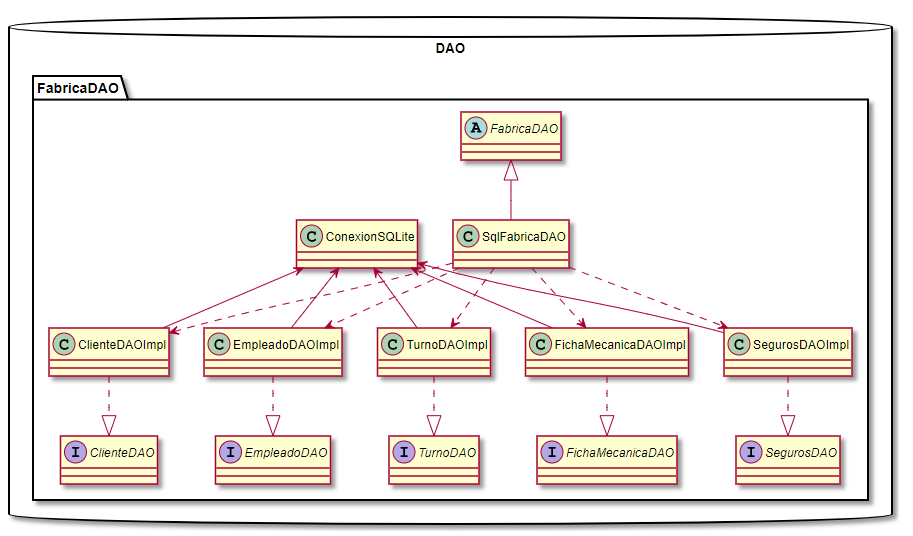
## Factory Method Persona



## DAO (Data access object) y DTO (Data transfer object)

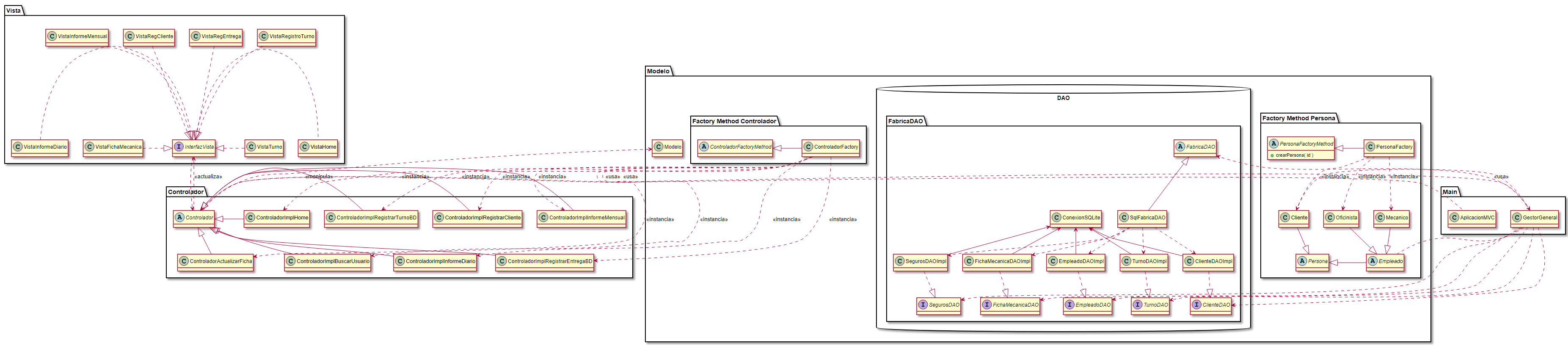
El patrón DAO propone separar por completo la lógica de negocio de la lógica para acceder a los datos, de esta forma, el DAO proporcionará los métodos necesarios para insertar, actualizar, borrar y consultar la información; por otra parte, la capa de negocio solo se preocupa por lógica de negocio y utiliza el DAO para interactuar con la fuente de datos.

Los DTO (Data Transfer Object) son utilizados por DAO para transportar los datos desde la base de datos hacia la capa de lógica de negocio y viceversa. Por ejemplo, cuando la capa de lógica de negocio llama al método create(), ¿qué es lo que hace DAO? inserta un nuevo dato… ¿pero qué dato? el que la capa de lógica de negocio le pase como parámetro… ¿y cómo se lo pasa este dato? bueno, a través de un DTO.



## MODELO-VISTA-CONTROLADOR (MVC)

Modelo-vista-controlador (MVC) es un patrón de arquitectura de software, que separa los datos y principalmente lo que es la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.​ Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.



I = INTERFAZ

A = CLASE ABSTRACTA

C = CLASE

## CASO DE USO

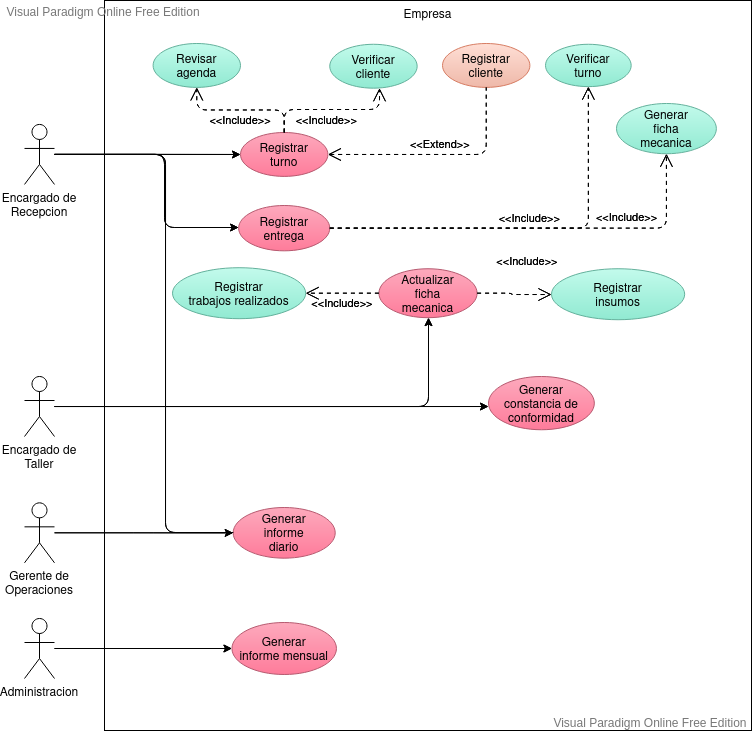
Un caso de uso es la descripción de una acción o actividad. ... Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas.

El Encargado cuenta con varias funciones, como por ejemplo registrar los turnos (Registrar Turno) .Debe revisar la agenda para buscar disponibilidad (Revisar Agenda), y también comprobar si la persona es cliente (Verificar Cliente), en caso de que no lo sea debe registrarla (Registrar Cliente).

El Encargado también debe registrar la entrega de un vehículo cuando el trabajo finaliza (Registrar Entrega). Debe revisar la agenda para verificar que el turno existe y tomar los datos para realizar la ficha mecánica (Generar Ficha Mecánica).

El encargado de taller colocará los trabajos realizados (Registrar Trabajos Realizados) y costos (Registrar Insumos) en dicha ficha mecánica (Actualizar Ficha Mecánica), para posteriormente finalizar la misma y generar la constancia de conformidad para el cliente(Generar Constancia de Conformidad).

Tanto el gerente de operaciones como el encargado realizarán un informe diario registrando todos los trabajos, las fichas mecánicas (Generar Informe Diario), para que posteriormente el administrador tenga mensualmente (Generar Informe Mensual) el informe de los trabajos realizados, compañías a cobrar, etc.



**DIAGRAMA BASE DE DATOS**

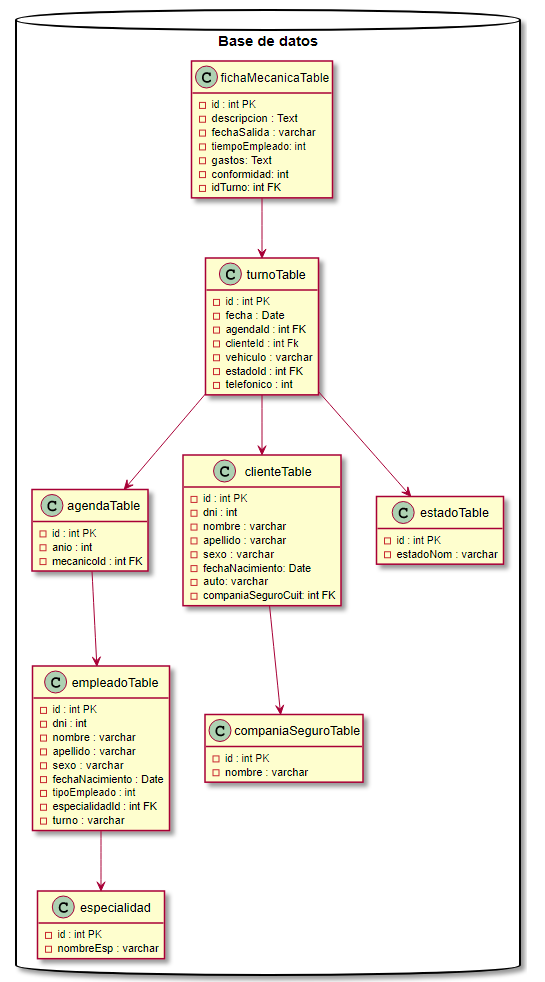
Es de vital importancia para cualquier programa tener un buen armado de las base de datos ya que son el corazón del programa. Hemos implementado tablas relacionadas para un correcto manejo de la información.

Cada ficha mecánica tendrá su turno, la cual cuenta con una agenda (mecánico y año), un cliente (datos personales) y un estado, que puede ser del 1 al 7 (mirar diagrama de estados para ver los siete estados).

A su vez cada empleado contará con una compañía de seguro.

La agenda tendrá un empleado que a su vez el mismo contará con una especialidad.

Hemos creado campos no usados en el programa con la idea de que se contemplen todos los cambios, o nuevas implementaciones, como por ejemplo ‘fecha de nacimiento’, ‘teléfono’, entre otros.



**TEST**

**Test unitarios**

Este tipo de testing consiste en probar de forma individual las funciones y/o métodos (de las clases, componentes y/o módulos que son usados por nuestro software).

Buenas pruebas unitarias no irían contra una base de datos, por ejemplo, sino que simularían la conexión.

Aquí nos interesa cómo funciona la unidad, no la interacción entre componentes (cosa que sería un test funcional o de integración).

Las ventajas de las pruebas unitarias es que por un lado los tests tardan menos tiempo en ejecutarse, por lo que se tiende a lanzarlos más a menudo. Además, las pruebas unitarias te fuerzan a escribir clases menos acopladas (con menos dependencias las unas de las otras), lo que mejora el diseño del software.

Si una prueba unitaria falla, sabes que es por un problema en el código.

**Test funcionales**

Las pruebas funcionales se centran en los requerimientos de negocio de una aplicación.

Estas pruebas verifican la salida (resultado) de una acción, sin prestar atención a los estados intermedios del sistema mientras se lleva a cabo la ejecución.

En este caso, el objetivo de las pruebas funcionales es comprobar que el software que se ha creado cumple con la función para la que se había pensado.

En este tipo de pruebas lo que miramos, lo que nos importan, son las entradas y salidas al software. Es decir, si ante una serie de entradas el software devuelve los resultados que nosotros esperábamos.

Aquí solo observamos que se cumpla la funcionalidad, no comprobamos que el software esté bien hecho, no miramos el diseño del software. Estudiamos el software desde la perspectiva del cliente, no del desarrollador.

Por eso este tipo de pruebas entran dentro de lo que se llaman pruebas de caja negra: aquí no nos centramos en cómo se generan las respuestas del sistema, solo analizamos los datos de entrada y los resultados obtenidos.

Una prueba funcional podría ser por ejemplo comprobar que en el formulario de mi página web si relleno un campo de teléfono con zzzz muestre un mensaje de campo no válido, y no me deje registrarme.

Hemos realizados tanto funcionales como unitarios, para esperar que el valor que ingresamos sea igual al valor devuelto, y para probar coincidencias con la base de datos a través de MOCK, que es un simulador, simulamos una conexión y búsqueda en nuestra base datos y comparamos valores ingresados a valores esperados.