Tema: II Proyecto programado

Código de la asignatura:

(ISB-21) Programación VI

Profesor:

Luis Daniel Vargas Gomes

Estudiantes:

Aleida Rivera Borge

Ronny Rafael Rodríguez Gomes

Sede universitaria a la que pertenece:

San Carlos

II Cuatrimestre

Año:

2025

Contenido

Resumen Ejecutivo

Este proyecto presenta el desarrollo de una aplicación de escritorio en Java, diseñada para optimizar la gestión operativa del taller mecánico. La solución incluye módulos funcionales que permiten registrar, consultar, modificar y eliminar información relacionada con Clientes, Vehículos, Servicios y Órdenes de Trabajo. Para garantizar la persistencia de datos de manera simple y eficiente, se emplean archivos de texto.

La interfaz gráfica fue construida con Swing, siguiendo un enfoque modular basado en CardLayout, lo que facilita la navegación dinámica entre paneles. La ventana principal gestiona múltiples vistas de manera centralizada y organizada, mientras que se integran submenús y formularios especializados para cada módulo, asegurando una experiencia de usuario clara y coherente.

Se incorpora además un panel de inicio neutral como base visual, junto con botones de cierre, validaciones mediante etiquetas visuales, y navegación por eventos, evitando recargas innecesarias. El diseño del sistema adopta principios de la arquitectura MVC, preparando el terreno para futuras mejoras.

Finalmente, el proyecto se documenta y versiona usando Git, lo que permite control colaborativo, trazabilidad de cambios y buenas prácticas en la organización del código. Esta solución representa un avance significativo en la digitalización del taller, mejorando la productividad, integridad de los datos y calidad del servicio ofrecido.

Introducción

El objetivo principal de este proyecto es construir un sistema que permita registrar, consultar, modificar y eliminar información relacionada con Clientes, Vehículos, Servicios y Órdenes de Trabajo, empleando archivos de texto como sistema de persistencia.

Desde el punto de vista visual, se implementó un diseño modular basado en CardLayout, lo que permite gestionar múltiples paneles de forma dinámica y organizada. La interfaz principal (Vista.java) incluye un panel central (PanelContenido) que muestra distintos

formularios y módulos según la acción del usuario. Además, se creó un Panel neutral de bienvenida (PanelInicio) que actúa como “base visual” al cerrar formularios o navegar entre secciones, mejorando la experiencia de usuario y evitando que los paneles queden visibles o superpuestos.

También se implementaron submenús dinámicos para cada módulo (Clientes, Vehículos, etc.), los cuales se despliegan al interactuar con los botones principales. Esto garantiza una estructura jerárquica clara, flexible y fácil de mantener.

Toda la navegación se gestiona mediante eventos, lo que permite alternar entre vistas sin recargar la ventana principal.

Por otro lado, el control de versiones se realiza mediante Git, permitiendo mantener un historial claro de cambios, colaboraciones y ramas. La presente documentación técnica se genera con enfoque modular y orientado a objetos, reflejando buenas prácticas en la organización de código, separación de responsabilidades y escalabilidad del sistema.

Objetivos General

Aplicar conceptos fundamentales de Java como clases, objetos, herencia, polimorfismo, interfaces y colecciones genéricas.

Utilizar Swing para la creación de una interfaz gráfica intuitiva y funcional.

Implementar persistencia de datos usando archivos de texto.

Organizar el desarrollo con control de versiones usando Git.

Documentar adecuadamente la arquitectura, funcionalidades y estructura del proyecto.

Objetivos específicos

Diseñar una ventana principal (Vista.java) con estructura modular, utilizando CardLayout para gestionar múltiples paneles de forma dinámica.

Implementar paneles de submenú que se desplieguen al interactuar con cada módulo (Clientes, Vehículos, etc.), permitiendo una navegación jerárquica ordenada.

Construir formularios para registrar, consultar, modificar y eliminar datos de clientes y vehículos, manteniendo consistencia visual y lógica interna.

Incorporar un panel de inicio (PanelInicio) como vista neutral, que se muestra al cerrar formularios o retornar al estado base de navegación.

Validar campos de entrada mediante etiquetas visuales y mensajes dinámicos, asegurando integridad de datos al registrar o modificar registros.

Estructurar paquetes que agrupen las clases según funcionalidad, promoviendo orden y escalabilidad del código fuente.

Garantizar la asociación entre clientes y vehículos mediante el uso de identificadores únicos (ID_Cliente, Placa), respetando las relaciones lógicas del sistema.

Gestionar versiones del proyecto de forma continúa utilizando Git, facilitando trazabilidad de cambios y control colaborativo del desarrollo.

Descripción del problema

El taller de servicios mecánico ha venido operando durante varios años utilizando métodos manuales para la gestión de sus operaciones diarias. Actualmente, el registro de clientes, vehículos, servicios prestados y facturación se realiza en hojas físicas o, en el mejor de los casos, en archivos de Excel sin un control centralizado. Este modelo de trabajo ha comenzado a mostrar serias limitaciones operativas y administrativas que afectan directamente la productividad del taller, la experiencia del cliente y el crecimiento del negocio.

Uno de los principales problemas es la pérdida frecuente de información, ya que no existe una base de datos formal. Es común que al momento de recibir un vehículo no se tenga acceso al historial de servicios anteriores, lo que retrasa la toma de decisiones técnicas y puede llevar a trabajos innecesarios o repetidos. Además, los datos de contacto de los clientes a menudo están incompletos o desactualizados, dificultando la comunicación efectiva sobre el estado de sus vehículos.

El manejo de las órdenes de trabajo también representa una debilidad. Estas se crean de forma manual y muchas veces no se siguen correctamente, lo que genera confusión sobre cuáles servicios ya se han realizado, cuáles están pendientes y cuáles están en espera de repuestos. Esto provoca retrasos, desorganización del personal y poca trazabilidad de los procesos.

La facturación es otro punto crítico. Al no contar con un sistema automatizado, los montos se calculan manualmente, lo que puede derivar en errores, cobros incorrectos y pérdida de confianza por parte de los clientes. Asimismo, el taller no puede generar reportes financieros ni operativos precisos, lo que impide evaluar el rendimiento mensual, controlar los ingresos o planificar mejoras a futuro.

Ante esta situación, es evidente la necesidad de implementar un sistema de gestión informático que permita automatizar y controlar todos los procesos del taller: desde el ingreso de clientes y vehículos, la creación y seguimiento de órdenes de servicio, hasta la facturación y generación de reportes. Con esta solución, se busca mejorar la eficiencia operativa, la atención al cliente y la toma de decisiones basada en datos reales y actualizados.

Desarrollo del sistema

Arquitectura General del Sistema

El sistema está estructurado en cuatro módulos principales:

- Gestión de Clientes
- Gestión de Vehículos
- Gestión de Servicios
- Gestión de Órdenes de Trabajo

Cada módulo está representado por clases específicas que manejan la lógica de datos, vistas Swing personalizadas y controladores encargados de los eventos de usuario.

Además, se emplea una entidad intermedia (Detalle_Servicio) que gestiona la relación muchos-a-muchos entre Servicios y Órdenes de Trabajo, sin embargo, esta anuente a cambios para mejores que se adapten a la funcionalidad del código, se puede crear una lista de objetos Detalle_Servicio (como clase auxiliar).

En cuanto a la persistencia de Datos

La persistencia se realiza mediante archivos de texto .txt. cada módulo tiene su archivo correspondiente, con estructura clara para facilitar lectura, escritura y modificación.

Las operaciones incluyen:

- Escritura secuencial de nuevos registros.
- Lectura completa para mostrar en la interfaz.
- Actualización de datos mediante reescritura del archivo.
- Eliminación lógica o física de líneas según el caso.

Relaciones entre Entidades

Relaciones Lógicas del Sistema:

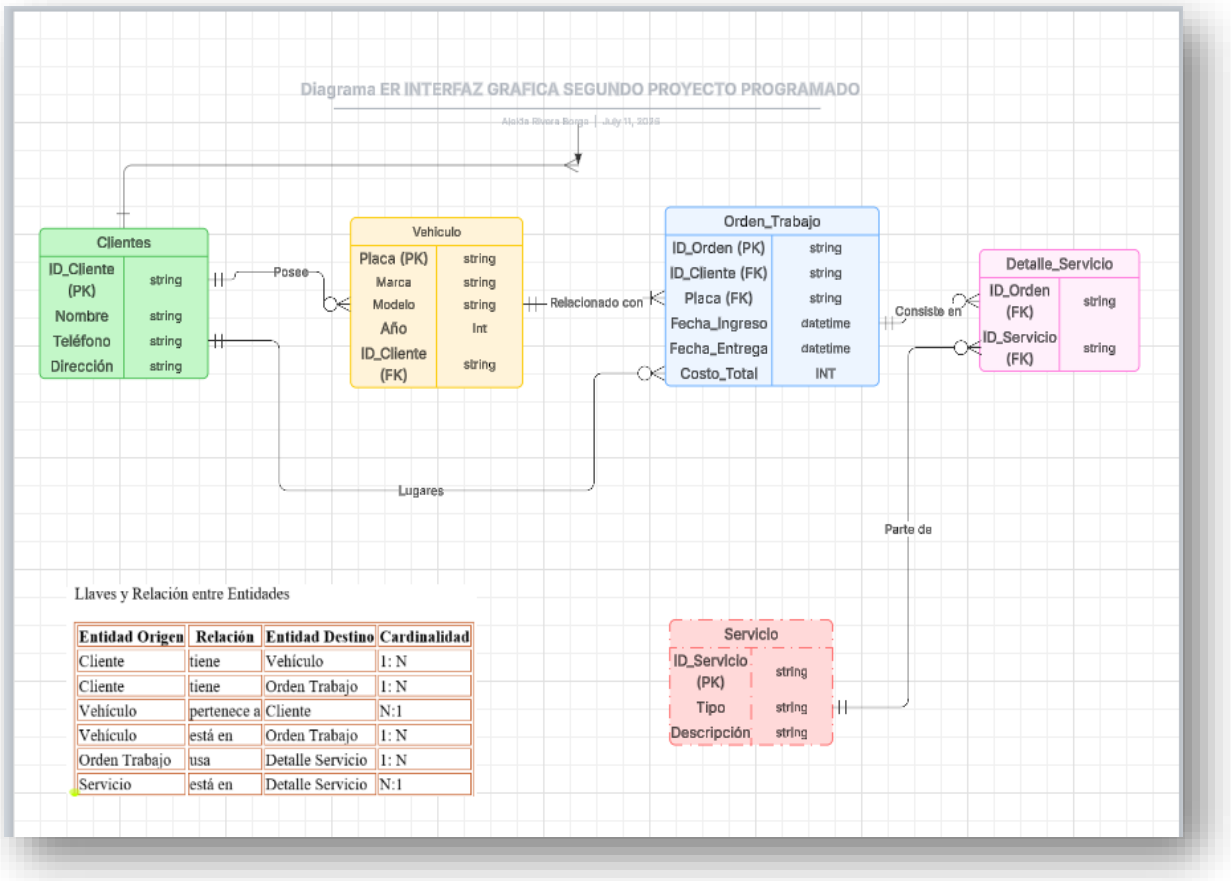
- Un Cliente puede tener múltiples Vehículos.
- Cada Vehículo está asociado a un único Cliente.
- Una Orden de Trabajo se vincula con un Cliente, un Vehículo y uno o más Servicios.
- La relación entre Órdenes y Servicios se maneja mediante una tabla intermedia (Detalle_Servicio).

Las llaves primarias (ID_Cliente, Placa, ID Servicio, ID_Orden) y llaves foráneas aseguran integridad en la vinculación de registros. Llaves y Relación entre Entidades

Entidad Origen	**Relación**	**Entidad Destino**	**Cardinalidad**
Cliente	tiene	Vehículo	1: N
Cliente	tiene	Orden Trabajo	1: N
Vehículo	pertenece a	Cliente	N:1
Vehículo	está en	Orden Trabajo	1: N
Orden Trabajo	usa	Detalle Servicio	1: N

Entidad Origen	Relación	Entidad Destino	Cardinalidad
Servicio	está en	Detalle Servicio	N:1

Diagrama Er Interfaz Gráfica Segundo Proyecto Programado



Interfaz Gráfica (Swing)

La interfaz se organiza en paneles por módulo, cada uno con:

- Formulario de entrada de datos (JTextField, JComboBox, JTextArea)

- Tabla para visualización (JTable)
- Botones de acción (JButton): Registrar, Consultar, Modificar, Eliminar
- Mensajes de estado (JLabel) y validaciones visuales
- Diseño limpio con layouts (BorderLayout, GridBagLayout, etc.) y estilos básicos

Cada vista permite interacción directa con el usuario, mostrando retroalimentación inmediata y permitiendo navegación entre módulos, dicha vista se pondrá a prueba para su aprobación, y esta anuente a cambios.

Implementación y pruebas Diseño del sistema
Elaboración de la interfaz (Vista)

Diagrama de flujo:

Gestión de Clientes

[Inicio: Módulo de Clientes]

↓

[Mostrar panel de opciones: Registrar, Consultar (Modificar, Eliminar)]

Registro de Cliente

[Usuario selecciona "Registrar"]

↓

[Ingreso de datos: Nombre, Teléfono, Dirección]

↓

[Validar campos vacíos o inválidos]

↓

¿Datos válidos?

→ No → [Mostrar error en JLabel]

→ Sí → [Guardar en archivo .txt + Mostrar confirmación]

Componentes:

btnRegistrar – botón para abrir el formulario de registro

btnConsultar – botón para abrir la tabla de clientes

Comportamiento esperado es cuando el usuario presiona Clientes, se activan los subbotones: Registrar y Consultar

Al presionar Registrar, se muestra el formulario de cliente (FormularioCliente.java) dentro de panelContenido (es un contenedor de otros paneles) usando CardLayout

Componentes del Formulario:

lblID – etiqueta para mostrar el ID generado es solo texto fijo por ahora

txtNombre – campo para nombre

txtTelefono – campo para teléfono

txtDireccion – campo para dirección

btnRegistrar – botón para guardar

lblMensaje – etiqueta vacía para mostrar mensajes de validación es solo texto fijo por ahora

¿Qué pasa después de llenar?

Una vez que el usuario llena los datos y hace clic en Registrar (botón), el comportamiento esperado es:

Validar que los campos no estén vacíos

Generar un ID si aplica.

Guardar los datos en archivo .txt.

Limpiar el formulario o mostrar un mensaje de "Cliente registrado".

Al presionar Consultar, se muestra la vista de clientes (ConsultarClientes.java) con una tabla

Desde esa tabla, se pueden realizar acciones como:

Modificar → cargar datos en un panel de edición

Eliminar → confirmar y borrar

Consulta de Clientes

[Usuario selecciona "Consultar"]

↓

[Leer archivo de clientes]

↓

[Mostrar lista en JTable]

Componentes:

tablaClientes – JTable (solo con columnas ID, Nombre, Teléfono, Dirección)

btnCerrar – para cerrar la ventana

Modificar y eliminar cliente desde la misma interfaz (ConsultaClientes.java)

La idea es seleccionar una fila en la tabla y luego:

Ver los datos seleccionados

Editarlos y guardar cambios

Eliminar ese cliente si desea

Ese formulario ConsultarClientes.java podría ser reutilizado también para:

Cargar los datos de un cliente ya existente para modificarlos

Pero para eso lo llamaría el botón Modificar, no el de Registrar.

Resultado final

ConsultaClientes.java Contiene:

Una tabla (tablaClientes)

Campos de texto que se llenarán al seleccionar una fila

Botones para modificar o eliminar ese cliente

Modificación de Cliente

[Usuario selecciona un cliente en JTable]

↓

[Mostrar datos actuales en formulario]

↓

[Modificar campos y confirmar cambios]

↓

[Actualizar archivo + Mostrar mensaje]

Eliminación de Cliente

[Seleccionar cliente desde JTable]

↓

[Confirmar eliminación]

↓

[Eliminar línea del archivo]

↓

[Actualizar vista + Mostrar confirmación]

Conexiones lógicas (PK y uso futuro de FK)

Cada cliente tendrá un ID_Cliente único.

Este ID será usado más adelante en los módulos de Vehículos y Órdenes como llave foránea, así que en esta vista debe generarse y visualizarse correctamente

Implementación para cerrar los formularios correctamente con CardLayout.

Se creo un panel neutral de inicio, este panel se utiliza como fondo “de descanso” cuando no se esté usando los formularios.

Se agrega botones “Cerrar” a tus formularios, también se verifica que todos los paneles estén registrados en Vista.java, la idea es desde el submenú de “Clientes”, navegar entre paneles.

Estos tres bloques están conectando distintos niveles de navegación:

btnClientes → despliega el submenú

btnRegistrarCliente y btnConsultarCliente → activan los formularios específicos

Todo funciona a través del CardLayout y su sistema de nombres ("registro", "consulta", "inicio"). Se implementará este enfoque en lo demás botones (vehículos, ordenes de trabajo y servicios)

Resultado

Al presionar “Cerrar” desde un formulario → vuelve al panel neutral

Al cambiar de módulo → no queda ningún panel viejo activo

La interfaz siempre se mantiene limpia y clara

El sistema Contiene:

- Panel neutral con mensaje de bienvenida
- Submenú que se despliega solo cuando se necesita
- Formularios que se abren y se cierran
- CardLayout que maneja todo como una caja invisible
- El sistema incluye una barra de navegación superior (JMenuBar).

Con el menú “Sistema”, diseñado para brindar control global al usuario sobre la aplicación. Dentro de este menú se incorporan dos elementos principales:

- JMenuItemSalir:
  Cierra completamente la aplicación mediante la acción programada que Invoca el método System.exit(0); Esto permite al usuario finalizar el sistema de forma directa desde cualquier módulo, sin necesidad de cerrar ventanas individuales.

- JMenuItemAcercade:
  Muestra información breve sobre el sistema, como el Contenido del mensaje "Sistema desarrollado por Aleida Rivera \nVersión 1.0\nGestión Taller", lo que nos da datos relevantes como autoría, versión y propósito del proyecto.

Registro de Vehículos

FormularioVehiculo.java

Este formulario es parte del módulo `moduloVehiculos` y está construido como un `JPanel` que se muestra dinámicamente dentro de la ventana principal del sistema (`Vista.java`).

Permite al usuario registrar nuevos vehículos asociados a clientes previamente creados. Su diseño facilita una entrada clara de información relevante, asegurando la integridad de los datos al momento de guardar.

Funcionalidad:

Interfaz tipo formulario con campos: ID Cliente, Marca, Modelo, Año y Placa.

Botón “Guardar Vehículo” para activar la lógica de validación.

Etiqueta de mensajes para mostrar retroalimentación visual de operaciones o errores.

Botón “Cerrar” que redirige al panel neutral (`PanelInicio`), manteniendo una navegación limpia mediante `CardLayout`.

Nota: Validación pendiente en campos de entrada, contemplada en fases posteriores de desarrollo.

ConsultaVehiculos.java

Gestión y Visualización

Forma parte del mismo módulo `moduloVehiculos`, y está orientado a la gestión visual y funcional de los datos vehiculares.

Este panel está diseñado para permitir al usuario consultar, modificar y eliminar vehículos previamente registrados en el sistema. Brinda una experiencia visual organizada para manipular registros sin complicaciones.

Funcionalidad:

Campo de búsqueda por `ID Cliente` para filtrar resultados en la tabla.

Tabla (`JTable`) con columnas como Marca, Modelo, Año, Placa e ID Cliente, que muestra todos los vehículos relacionados.

Panel de edición con campos para modificar los datos del vehículo seleccionado en la tabla.

Botones de acción:

Modificar para actualizar información existente

Eliminar para borrar el registro seleccionado.

Cerrar para regresar al panel neutral y limpiar la vista.

Uso de `CardLayout` para transición fluida entre vistas, manteniendo coherencia visual en todo momento.

Módulo de Vehículos

↓

[Mostrar opciones: Registrar, Consultar, Modificar, Eliminar]

Registro de vehículo

[Usuario selecciona "Registrar"]

↓

[Buscar o seleccionar Cliente (por ID_Cliente o nombre)]

↓

¿Cliente existe?

→ No → [Mostrar error: “Cliente no registrado”]

→ Sí → [Ingresar datos del vehículo: Marca, Modelo, Año, Placa]

↓

[Validar campos]

↓

¿Datos válidos?

→ No → [Mostrar error en JLabel]

→ Sí → [Guardar vehículo en archivo + Asociar con ID_Cliente]

↓

[Mostrar confirmación en JLabel]

Consulta de vehículos por cliente

[Usuario selecciona "Consultar"]

↓

[Buscar cliente (ID o nombre)]

↓

[Leer archivo y filtrar vehículos del cliente]

↓

[Mostrar lista en JTable]

Modificación de vehículo

[Seleccionar vehículo desde la lista]

↓

[Mostrar datos actuales en formulario]

↓

[Modificar campos deseados]

↓

[Guardar cambios en archivo + Mostrar confirmación]

Eliminación de vehículo

[Seleccionar vehículo en la tabla]

↓

[Confirmar eliminación]

↓

[Eliminar línea del archivo]

↓

[Actualizar vista + Mostrar mensaje de éxito]

Relaciones lógicas clave

Cada vehículo está vinculado a un Cliente por ID_Cliente (llave foránea).

La Placa del vehículo actúa como su llave primaria única.

Una vez registrado, el vehículo puede usarse luego en Órdenes de Trabajo → este vínculo debe conservarse para futura conexión.

Gestión de Servicios

Paquete modulo Servicios:

contiene todas las clases relacionadas al módulo de servicios.

FormularioServicio.java (Registro de Servicio)

Componentes que nos ayudan a entender la lógica de este programa:

Validar campos obligatorios para un mejor desarrollo del proceso

Generar un ID_Servicio automático

Guardar en archivo .txt

Mostrar mensaje en lblMensaje

Botón “Cerrar” vuelve al panel "inicio"

La estructura incluye:

Un combo desplegable con tipos de servicio predefinidos (Mecánica, Pintura, Revisión, Otros)

Un área de texto amplia para la descripción

Otro combo para prioridad (Alta, Media, Baja)

Dos RadioButton agrupados para estado: Activo / Inactivo

Botones “Guardar Servicio” y “Cerrar”

Un mensaje al pie del formulario para mostrar confirmación o errores

Arriba: selección de orden

Medio: tabla con selección múltiple

Abajo: botones + confirmación

Todo en un panel bien dividido por BorderLayout

AsociarServicioOrden.java (Vincular servicio a orden)

Componentes que nos ayudan a entender la lógica de este programa:

Cargar lista de órdenes y servicios

Permitir selección múltiple (relación muchos-a-muchos)

Guardar asociaciones en entidad Detalle_Servicio

Mostrar mensaje de éxito al finalizar

Estructura de Orden servicio nos ayuda a comprender qué debe hacer cada sección.

Selección de orden: elegir sobre cuál trabajar

Tabla: elegir qué servicios aplicar a esa orden

Botón “Asociar”: realiza la vinculación (más adelante)

Botón “Cerrar”: vuelve al panel "inicio" con CardLayout

Gestión de Servicios

Módulo de Servicios

↓

[Mostrar opciones: Registrar Servicio, Asociar Servicio a Orden]

Registro de Servicio

[Seleccionar "Registrar"]

↓

[Ingresar datos: Tipo, Descripción]

↓

[Validar campos vacíos]

↓

¿Datos válidos?

→ No → [Mostrar mensaje de error en JLabel]

→ Sí → [Guardar en archivo .txt + Mostrar mensaje de confirmación]

Asociación de Servicios a Órdenes

[Seleccionar "Asociar a orden"]

↓

[Mostrar lista de órdenes de trabajo]

↓

[Seleccionar orden específica]

↓

[Mostrar lista de servicios disponibles]

↓

[Seleccionar uno o varios servicios]

↓

[Guardar asociación en entidad intermedia Detalle_Servicio]

↓

[Mostrar confirmación de vinculación]

Aspectos lógicos importantes

Cada servicio tiene un ID_Servicio único (llave primaria).

Se permite relación muchos-a-muchos entre servicios y órdenes → por eso se pretende usar Detalle_Servicio.

Módulo: Órdenes de Trabajo

El módulo de Órdenes de Trabajo forma parte del sistema de gestión académica y permite administrar los registros relacionados con servicios realizados a vehículos. Se accede a de igual manera a través del menú lateral principal

Al tocar el botón "Órdenes de Trabajo", se despliega un submenú que contiene dos accesos funcionales:

Registrar orden

Dentro del formulario se agrupan las siguientes secciones:

Cliente: permite seleccionar o validar el usuario relacionado con la orden.

Vehículo asociado: filtrado según el cliente, vincula el servicio a un vehículo específico.

Servicios realizados: ofrece una lista desplegable de servicios disponibles y la posibilidad de agregarlos dinámicamente a una tabla, para que facilite la visualización y gestión de los servicios que se incluye en la orden.

Fechas y estado: sección para ingresar fecha de ingreso y entrega.

Costo total: la idea funcional es que se calcule automáticamente según los servicios registrados.

Acción principal: Registrar orden sería el botón que valida y guarda la orden, mostrando una confirmación visual.

¿Para qué sirve esta información?

Con la finalidad de crear una nueva orden, se aplica la selección de cliente y vehículo asociado mediante la adición dinámica de servicios, se implantará el cálculo automático del costo total para así obtener el registro final de la orden con confirmación visual

Todo eso se asocia con el botón "Registrar orden", donde se recopilan y procesan los datos para guardar una nueva entrada en el sistema.

Esta interacción entre los botones permite que la experiencia de usuario sea fluida y profesional. El sistema asegura que la vista correspondiente se muestre al tocar cada botón, y que las secciones dentro del formulario respondan lógicamente según los datos proporcionados.

Panel ConsultaOrdenTrabajo

El panel tiene como propósito permitir al usuario:

Buscar órdenes de trabajo existentes aplicando filtros por cliente, vehículo, fecha y estado y con esto visualizar los resultados en una tabla ordenada, consultar detalles completos de una orden específica y finamente ejecutar acciones como modificar o cerrar la orden seleccionada.

Estructura visual del formulario

1. Panel de filtros de búsqueda permite filtrar órdenes por:

- Cliente (txtFiltroCliente)
- Vehículo (txtFiltroVehiculo)
- Fecha ingreso (txtFiltroFecha)
- Estado (cbFiltroEstado)
- Botón de acción: btnBuscarOrdenes

Al presionar “Buscar”, se actualiza la tabla con las órdenes que cumplan con los criterios.

2. Tabla de resultados (tablaOrdenes)

- Muestra órdenes filtradas con columnas: ID, Cliente, Vehículo, Fechas, Estado, Costo total
- Está integrada en un JScrollPane para facilitar la navegación al seleccionar una fila, sus datos se cargan en el panel inferior para visualizar y gestionar la orden.

3. Panel de detalle de la orden seleccionada

Campos visibles y/o editables:

- Cliente (txtDetalleCliente) – no editable ya que este lo debe jalar.
- Vehículo (cbDetalleVehiculo)
- Fecha Ingreso (txtDetalleIngreso)
- Fecha Entrega (txtDetalleEntrega)
- Estado (txtDetalleEstado) – informativo
- Costo Total (txtDetalleCosto)
- Servicios (tablaDetalleServicios)

La idea es brindar una vista completa de la orden seleccionada desde la tabla que permita realizar acciones si la orden está abierta.

4. Botones de acción

- Modificar orden (btnModificarOrden): permite editar los datos si la orden está abierta
- Cerrar orden (btnCerrarOrden): cambia el estado a “Cerrada” y bloquea ediciones futuras
- lblMensajeAccion: muestra confirmaciones o errores durante las operaciones

Relaciones clave del sistema

- ID_Cliente: conecta la orden con el cliente
- Placa: vincula el vehículo involucrado
- ID_Servicio: servicios relacionados se gestionan como una lista de objetos Detalle_Servicio

Comportamiento esperado

Al ingresar al panel, el usuario puede:

- Filtrar órdenes, ver resultados en la tabla
- Seleccionar una orden y así consultar su información completa

- Modificar o cerrar la orden según su estado

Órdenes de Trabajo

Módulo de Órdenes de Trabajo

↓

[Mostrar opciones: Crear, Consultar, Modificar, Cerrar Orden]

Crear una Orden de Trabajo

[Seleccionar "Crear Orden"]

↓

[Seleccionar Cliente (por ID o nombre)]

↓

¿Cliente existe?

→ No → [Mostrar error: “Cliente no registrado”]

→ Sí → [Seleccionar Vehículo asociado]

↓

¿Vehículo válido?

→ No → [Mostrar error]

→ Sí → [Seleccionar uno o varios Servicios]

↓

[Ingresar fechas y costo total]

↓

[Validar datos]

↓

¿Datos válidos?

→ No → [Mostrar error]

→ Sí → [Guardar orden en archivo + Crear vínculo con Detalle_Servicio]

↓

[Mostrar confirmación de creación]

Consultar órdenes de trabajo

[Seleccionar "Consultar"]

↓

[Buscar por cliente / fecha / vehículo / estado]

↓

[Mostrar lista de órdenes en JTable]

Modificar orden de trabajo

[Seleccionar orden desde la lista]

↓

[Mostrar datos actuales en formulario]

↓

[Editar fechas, servicios, vehículo si aplica]

↓

[Guardar cambios + actualizar archivo]

↓

[Mostrar mensaje de éxito]

Cerrar una orden

[Seleccionar orden en JTable]

↓

[Confirmar cierre]

↓

[Cambiar estado de la orden → “Cerrada”]

↓

[Guardar cambio + mostrar confirmación]

Relaciones clave que se reflejan en esta vista

La orden se vincula con:

ID_Cliente (desde Cliente)

Placa (desde Vehículo)

ID_Servicio (desde entidad intermedia Detalle_Servicio, en tema de código se puede crear una lista de objetos Detalle_Servicio (como clase auxiliar).

En la vista Swing, esto implica usar:

Selectores (JComboBox) y tablas (JTable) para elegir entidades.

Formularios bien organizados para entrada de datos.

Validaciones con mensajes visuales (JLabel, ToolTip)

Y los archivos .txt/ para guardar cada parte vinculada.

Implementación y pruebas

Control de Versiones

Se emplea Git para:

- Seguimiento de cambios en el código fuente
- Manejo de ramas para nuevas funcionalidades;
- Control de versiones estable en cada avance
- Documentación de commits significativa

Conclusiones

El proyecto refleja una correcta aplicación de los principios de la programación orientada a objetos, con enfoque modular y funcional. La interfaz gráfica fue desarrollada para facilitar la experiencia del usuario, mientras que la persistencia mediante archivos brinda una solución accesible para el manejo de datos sin base de datos.

Además, se implementó una estructura modular con navegación dinámica utilizando CardLayout, permitiendo que los formularios se visualicen, oculten o alternen sin perder fluidez. El uso de paneles neutros, submenús desplegables y botones inteligentes de cierre contribuyen a una experiencia visual limpia y profesional.

La organización del código en paquetes temáticos como modulo Clientes y modulo Vehículos promueve el orden y escalabilidad del sistema, mientras que la separación lógica de formularios y modelos refuerza el enfoque orientado a objetos. El sistema

también demuestra buenas prácticas en el manejo de eventos, validación de datos, reutilización de componentes, y diseño limpio de interfaz.

Finalmente, la gestión del proyecto con Git y la elaboración de documentación técnica permiten mantener trazabilidad, mejora continua y claridad en cada etapa del desarrollo. El sistema queda listo para ser extendido incluso para escalar hacia persistencia con base de datos o conexión en red en versiones futuras.

Recomendaciones

Estas serían algunas de las recomendaciones de implantación a fututo:

Integrar base de datos relacional:

Para mejorar la persistencia, sería recomendable migrar de archivos .txt a una base de datos SQL (como MySQL o PostgreSQL), permitiendo operaciones más robustas, filtros complejos, y relaciones entre entidades.

Agregar funciones de búsqueda avanzada:

Permitir filtros dinámicos, búsqueda por múltiples campos, y ordenamientos dentro de tablas interactivas.

Internacionalización y personalización:

Permitir que el sistema soporte varios idiomas o configuraciones regionales para adaptarlo a diferentes entornos de trabajo.

Integrar logs o sistema de auditoría:

Para registrar acciones del usuario, como modificaciones y eliminaciones, brindando trazabilidad adicional.

URL del repositorio

https://github.com/AleidaRivera/ProyectoII

Referencias: bibliográficas

1. Oracle. (s.f.). *Scanning using java.util.Scanner*. Oracle. Recuperado de https://docs.oracle.com/javase/tutorial/essential/io/scanning.html
2. W3Schools. (s.f.). *Java Try and Catch*. Recuperado de https://www.w3schools.com/java/java_try_catch.asp
3. Oracle. (s.f.). *How to use tables*. Oracle. Recuperado de https://docs.oracle.com/javase/tutorial/uiswing/components/table.html
4. Stack Overflow. (s.f.). *Resultados de búsqueda: java read text file and populate JTable*. Recuperado de https://stackoverflow.com/search?q=java+read+text+file+and+populate+jtable
5. GeeksforGeeks. (s.f.). *Customizing JTable columns in Java*. Recuperado de https://www.geeksforgeeks.org/customizing-jtable-columns-in-java/
6. GeeksforGeeks. (s.f.). *Different ways of reading a text file in Java*. Recuperado de https://www.geeksforgeeks.org/java/different-ways-reading-text-file-java/
7. GeeksforGeeks. (s.f.). *NumberFormatException in Java with examples*. Recuperado de https://www.geeksforgeeks.org/java/numberformatexception-in-java-with-examples/
8. Oracle. (s.f.). *JOptionPane (Java Platform SE 8)*. Recuperado de https://docs.oracle.com/javase/8/docs/api/javax/swing/JOptionPane.html
9. Lucidchart. (s.f.). *Diagrama EER en Lucidchart*. Recuperado de https://lucid.app/lucidchart/78ba35f3-5f89-4728-bc7c-7928112db8ab/edit?page=0_0&invitationId=inv_08121f2d-35dd-469b-af0d-c051ba69e917