Fecha de entrega: 6 de agosto del 2022 Hora de entrega: 7:00 pm

II Proyecto Programado

(ISB-21) PROGRAMACIÓN IV Profesor: Luis Daniel Vargas Gómez Nombre de los autores: Aleida Rivera Borge y Ronny Rafael Rodríguez Gómez Universidad Internacional San Isidro Labrador

Pag 2 Tabla de contenido

Pag1 Portada

Pag3 Resumen ejecutivo

Pag4 Introducción

5. Objetivo general

6 Objetivo específicos

7 Descripción del problema

8 Desarrollo del sistema

Diseño del sistema

Arquitectura General del Sistema

. Implementación y pruebas

Pag 12 conclusiones

10 recomendaciones

Pag Bibliografía (Referencias en formato APA)

Nota: Este documento está sujeto a cambios

Resumen ejecutivo

Introducción

Este proyecto consiste en el desarrollo de una aplicación de escritorio en Java, utilizando la biblioteca Swing para la creación de interfaces gráficas. El objetivo principal 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 (.txt) 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. Además, se incorporaron controles como botones de cierre en cada formulario, validaciones visuales con etiquetas (JLabel), y una base para una arquitectura tipo MVC en crecimiento.

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.

5 Objetivo general

Objetivos del Proyecto

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.

6 Objetivo específicos

Objetivos del Proyecto

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 (JLabel) 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.

7 Descripción del problema

8 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.

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 |
| Servicio | está en | Detalle Servicio | N:1 |
|  |  |  |  |

## Diagrama Er Interfaz Gráfica Segundo Proyecto Programado

Diagrama

El contenido generado por IA puede ser incorrecto.

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.

# Elaboración de la interfaz (Vista)

Diagrama de flujo: Gestión de Clientes (versión estructurada)

Inicio del Módulo

[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".

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

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

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Anexo

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.

**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]

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

**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").

La idea es implementar 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

Anexo

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Control de Versiones

Se emplea Git para:

* Seguimiento de cambios en el código fuente: En espera del Codigo
* Manejo de ramas para nuevas funcionalidades; Detallar la rama
* 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 moduloClientes y moduloVehiculos 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.

10 recomendaciones

Estas serian 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.

Pag Bibliografía (Referencias en formato APA)

[https://lucid.app/lucidchart/78ba35f3-5f89-4728-bc7c-7928112db8ab/edit?page=0\_0&invitationId=inv\_08121f2d-35dd-469b-af0d-c051ba69e917#](https://lucid.app/lucidchart/78ba35f3-5f89-4728-bc7c-7928112db8ab/edit?page=0_0&invitationId=inv_08121f2d-35dd-469b-af0d-c051ba69e917)