Universidad de San Carlos de Guatemala
Facultad de Ingeniería

PROYECTO #2
MANUAL TÉCNICO
TALLER MECANICO

Introducción a la Programación 1

Auxiliar Sebastian Alejandro Velasquez Bonilla

Guatemala, abril de 2025

Manual Técnico - USAC Taller Automotriz

Plataforma de Ejecución

- **Lenguaje de Programación**: Java
- **JDK**: 17 o superior
- **IDE recomendado**: NetBeans
- **Sistema Operativo**: Multiplataforma (Windows, Linux, macOS)
- **Memoria recomendada**: 4GB mínimo
- **Requerimientos adicionales**: Java Runtime Environment (JRE)

Arquitectura del Sistema

El sistema está implementado siguiendo el patrón arquitectónico MVC (Modelo-Vista-Controlador), permitiendo una separación clara entre los datos, la interfaz de usuario y la lógica del negocio.

Diccionario de Métodos

Gestión de Clientes

- `ordenarClientesPorDPI(Cliente[] clientes)`: Ordena el arreglo de clientes utilizando el método de burbuja de forma ascendente según el identificador (DPI).
- `buscarCliente(String idCliente)`: Busca un cliente específico por su identificador y retorna el objeto Cliente si lo encuentra.
- `agregarCliente(Cliente cliente)`: Añade un nuevo cliente al sistema validando que el identificador no exista previamente.
- `modificarCliente(Cliente cliente)`: Actualiza los datos de un cliente existente en el sistema.
- `eliminarCliente(String idCliente)`: Elimina un cliente del sistema utilizando su identificador como criterio de búsqueda.
- `esTipoOro(Cliente cliente)`: Verifica si un cliente cumple con los requisitos para ser considerado cliente oro según el número de servicios realizados.
- `actualizarTipoCliente(Cliente cliente)`: Actualiza el tipo de cliente (normal u oro) según la cantidad de servicios que ha realizado.

Gestión de Automóviles

- `ordenarAutomovilesAscendente(Automovil[] automoviles)`: Ordena los automóviles por placa en orden ascendente usando el algoritmo ShellSort.

- `ordenarAutomovilesDescendente(Automovil[] automoviles)`: Ordena los automóviles por placa en orden descendente usando el algoritmo ShellSort.
- `agregarAutomovil(Cliente cliente, Automovil automovil)`: Agrega un automóvil a la lista de vehículos de un cliente específico.
- `buscarAutomovil(String placa)`: Busca un automóvil en el sistema por su número de placa.
- `modificarAutomovil(Cliente cliente, Automovil automovil)`: Actualiza los datos de un automóvil existente de un cliente específico.
- `eliminarAutomovil(Cliente cliente, String placa)`: Elimina un automóvil de la lista de vehículos de un cliente utilizando la placa como criterio.

Gestión de Repuestos

- `agregarRepuesto(Repuesto repuesto)`: Añade un nuevo repuesto al inventario del taller.
- `modificarRepuesto(Repuesto repuesto)`: Actualiza la información de un repuesto existente en el inventario.
- `eliminarRepuesto(int idRepuesto)`: Elimina un repuesto del inventario utilizando su identificador.
- `buscarRepuesto(int idRepuesto)`: Busca un repuesto específico por su identificador y lo retorna si existe.
- `reducirExistencias(Repuesto repuesto, int cantidad)`: Reduce la cantidad disponible de un repuesto específico cuando se utiliza en un servicio.
- `verificarExistencias(Repuesto repuesto, int cantidad)`: Verifica si hay suficientes unidades de un repuesto para realizar un servicio.
- `obtenerRepuestosMasUsados(int cantidad)`: Retorna una lista con los repuestos más utilizados en los servicios del taller.
- `obtenerRepuestosCaros(int cantidad)`: Retorna una lista con los repuestos más caros del inventario.

Gestión de Servicios

- `agregarServicio(Servicio servicio)`: Añade un nuevo servicio al catálogo del taller.
- `modificarServicio(Servicio servicio)`: Actualiza la información de un servicio existente.
- `eliminarServicio(int idServicio)`: Elimina un servicio del catálogo utilizando su identificador.
- `buscarServicio(int idServicio)`: Busca un servicio específico por su identificador y lo retorna si existe.
- `calcularPrecioTotal(Servicio servicio)`: Calcula el precio total de un servicio sumando el costo de mano de obra y el de los repuestos.
- `agregarRepuestoAServicio(Servicio servicio, Repuesto repuesto)`: Agrega un repuesto a la lista de repuestos necesarios para un servicio específico.

- `eliminarRepuestoDeServicio(Servicio servicio, int idRepuesto)`: Elimina un repuesto de la lista de repuestos de un servicio.
- `esCompatibleCon(Servicio servicio, Automovil automovil)`: Verifica si un servicio es compatible con un automóvil comparando marca y modelo.
- `obtenerServiciosMasUsados(int cantidad)`: Retorna los servicios más solicitados en el taller.
- `diagnosticarServicio(Automovil automovil)`: Selecciona aleatoriamente un servicio compatible con el automóvil especificado para diagnóstico.

Gestión de Órdenes de Trabajo

- `generarOrdenTrabajo(Automovil automovil, Cliente cliente, Servicio servicio, Empleado empleado)`: Crea una nueva orden de trabajo y la asigna a un empleado.
- `actualizarEstadoOrden(OrdenTrabajo orden, String nuevoEstado)`: Actualiza el estado de una orden de trabajo (en espera, en servicio, listo).
- `obtenerOrdenesEnEspera()`: Retorna todas las órdenes de trabajo en estado de espera.
- `obtenerOrdenesEnProceso()`: Retorna todas las órdenes de trabajo en estado de proceso.
- `obtenerOrdenesFinalizadas()`: Retorna todas las órdenes de trabajo finalizadas.
- `ingresarAutomovilACola(OrdenTrabajo orden)`: Agrega una orden de trabajo a la cola de espera para ser procesada.
- `asignarAutomovilAMecanico(OrdenTrabajo orden, Empleado mecanico)`: Asigna un automóvil a un mecánico disponible para realizar el servicio.
- `finalizarServicio(OrdenTrabajo orden)`: Marca una orden de trabajo como finalizada y libera al mecánico asignado.

Gestión de Facturas

- `generarFactura(OrdenTrabajo orden)`: Crea una nueva factura a partir de una orden de trabajo finalizada.
- `pagarFactura(Factura factura)`: Marca una factura como pagada y actualiza el contador de servicios del cliente.
- `obtenerFacturasPendientes(Cliente cliente)`: Retorna todas las facturas pendientes de pago de un cliente específico.
- `obtenerClientesConFacturasPendientes()`: Retorna todos los clientes que tienen facturas pendientes de pago.
- `generarPDF(Factura factura, String rutaDestino)`: Genera un archivo PDF con los detalles de la factura.

Manejo de Archivos

- `cargarRepuestosDesdeArchivo(String rutaArchivo)`: Lee un archivo de extensión .tmr y carga los repuestos al sistema.
- `cargarServiciosDesdeArchivo(String rutaArchivo)`: Lee un archivo de extensión .tms y carga los servicios al sistema.
- `cargarClientesAutomovilesDesdeArchivo(String rutaArchivo)`: Lee un archivo de extensión .tmca y carga los clientes con sus automóviles al sistema.
- `serializarDatos(Object datos, String rutaArchivo)`: Guarda los datos del sistema en un archivo binario mediante serialización.
- `deserializarDatos(String rutaArchivo)`: Carga los datos del sistema desde un archivo binario previamente serializado.

Generación de Reportes

- `generarReporteClientes(String rutaDestino)`: Genera un reporte PDF con la información de todos los clientes, separados por tipo (oro y normal).
- `generarGraficaClientesPorTipo()`: Crea una gráfica de pastel que muestra la cantidad de clientes oro versus clientes normales.
- `generarReporteTopRepuestosUsados(String rutaDestino)`: Genera un reporte PDF con los 10 repuestos más utilizados en el taller.
- `generarGraficaTopRepuestosUsados()`: Crea una gráfica de barras mostrando los 10 repuestos más utilizados.
- `generarReporteTopRepuestosCaros(String rutaDestino)`: Genera un reporte PDF con los 10 repuestos más caros del inventario.
- `generarGraficaTopRepuestosCaros()`: Crea una gráfica de barras mostrando los 10 repuestos más caros.
- `generarReporteTopServicios(String rutaDestino)`: Genera un reporte PDF con los 10 servicios más solicitados en el taller.
- `generarGraficaTopServicios()`: Crea una gráfica de barras mostrando los 10 servicios más solicitados.
- `generarReporteAutomovilesRepetidos(String rutaDestino)`: Genera un reporte PDF con los 5 modelos de automóviles que más se repiten entre los clientes.
- `generarGraficaAutomovilesRepetidos()`: Crea una gráfica comparativa entre los dos modelos de automóviles más repetidos.

Gestión de Hilos

- `iniciarProcesoAutomovil(OrdenTrabajo orden)`: Inicia un hilo que simula el proceso de un automóvil en el taller según los tiempos establecidos.
- `procesarEnEspera(OrdenTrabajo orden, int tiempoEspera)`: Simula el tiempo que un automóvil está en la cola de espera.
- `procesarEnServicio(OrdenTrabajo orden, int tiempoServicio)`: Simula el tiempo que toma realizar el servicio al automóvil.
- `procesarListo(OrdenTrabajo orden, int tiempoListo)`: Simula el tiempo que un automóvil está listo esperando ser retirado.

- `actualizarInterfazProgreso(JTable tablaEspera, JTable tablaEnProceso, JTable tablaListos)`: Actualiza las tablas de la interfaz que muestran el progreso de los automóviles en el taller.

Validaciones

- `esNumerico(String texto)`: Verifica si una cadena de texto contiene únicamente caracteres numéricos.
- `soloLetras(String texto)`: Verifica si una cadena de texto contiene únicamente letras.
- `campoVacio(String campo)`: Verifica si un campo está vacío o contiene solo espacios en blanco.
- `verificarCredenciales(String usuario, String contraseña)`: Verifica si las credenciales de acceso son correctas.
- `obtenerTipoUsuario(String usuario)`: Determina el tipo de usuario (administrador, cliente, mecánico) a partir del nombre de usuario.
- `verificarCompatibilidad(String marcaServicio, String modeloServicio, String marcaAuto, String modeloAuto)`: Verifica la compatibilidad entre un servicio y un automóvil basado en marca y modelo.

Control de Bitácora

- `registrarEvento(String usuario, String accion, boolean resultado, String detalles)`: Registra un evento en la bitácora del sistema.
- `mostrarBitacora()`: Muestra la bitácora de eventos del sistema en una ventana.
- `limpiarBitacora()`: Limpia todos los registros de la bitácora al finalizar la ejecución del programa.

Consideraciones Técnicas Adicionales

1. **Serialización**: Todos los objetos del modelo implementan la interfaz `Serializable` para garantizar la persistencia de datos entre sesiones.

2. **Hilos**: Se utilizan hilos para simular el flujo de trabajo del taller, permitiendo que múltiples vehículos sean procesados simultáneamente y que la interfaz se actualice en tiempo real.

3. **Ordenamientos**: Se implementan algoritmos de ordenamiento personalizados (Burbuja para clientes y ShellSort para automóviles) según los requerimientos del proyecto.

4. **Cola de prioridad**: Los clientes tipo "oro" tienen prioridad en la cola de espera, lo que se implementa mediante una verificación del tipo de cliente al encolar un nuevo vehículo.

5. **Gestión de archivos**: El sistema puede leer archivos con extensiones específicas (.tmr, .tms, .tmca) para la carga masiva de datos.

6. **Seguridad**: Se implementa un sistema básico de autenticación para diferenciar entre administradores, clientes y mecánicos.

7. **Reportes**: Los reportes se generan en formato PDF y incluyen gráficas generadas con la biblioteca JFreeChart.

8. **Interfaz gráfica**: Se utiliza Swing para la implementación de todas las interfaces de usuario, organizadas según el patrón MVC.

9. **Validaciones**: Todas las entradas de usuario son validadas para garantizar la integridad de los datos en el sistema.

10. **Compatibilidad**: Se verifica la compatibilidad entre servicios y vehículos según marca y modelo antes de procesar una orden de trabajo.