Universidad de San Carlos de Guatemala
Escuela de Ciencias y Sistemas
Facultad de Ingeniería
Introducción a la Programación y Computación 1
Escuela de Vacaciones junio 2025
Catedrático: Ing. Neftalí de Jesús Calderón Mendez
Tutor académico: Anthony Alexander Aquino Santiago

Manual técnico

Josue Geovany Yahir Perez Avendaño
202300814
1/07/2025

Plataforma de Ejecución y Requisitos

Requisitos Mínimos del Sistema

Sistema Operativo:

- Windows 10 o superior
- macOS 10.14 o superior
- Linux (Ubuntu 18.04 o distribuciones equivalentes)

Hardware:

- Procesador: Intel Core i3 o AMD equivalente (mínimo 2.0 GHz)
- RAM: 4 GB mínimo, 8 GB recomendado
- Espacio en Disco: 500 MB libres para la aplicación
- Resolución de Pantalla: 1024x768 mínimo, 1920x1080 recomendado

Software Requerido:

- Java Runtime Environment (JRE): Versión 8 o superior
- Java Development Kit (JDK): Versión 8 o superior (para desarrollo)

Dependencias Externas

iText: Para generación de reportes PDF

JFreeChart: Para generación de gráficas

Swing: Para interfaces gráficas (incluido en JDK)

Arquitectura del Sistema

El sistema implementa el patrón Modelo-Vista-Controlador (MVC) con las siguientes capas:

- Modelo (Model): Clases de datos y lógica de negocio
- Vista (View): Interfaces gráficas de usuario

- Controlador (Controller): Lógica de control y coordinación

Diccionario de Métodos

Clase: Ingrediente

Propósito: Representa un ingrediente del inventario del restaurante.

Métodos Principales:

public Ingrediente(int id, String nombre, String marca, int existencias, String unidades, double precio)

- **Descripción:** Constructor que inicializa un ingrediente con todos sus atributos.
- **Parámetros:**
    - id: Identificador único del ingrediente
    - nombre: Nombre del ingrediente
    - marca: Marca comercial del ingrediente
    - existencias: Cantidad disponible en inventario
    - unidades: Unidad de medida (sobres, paquetes, etc.)
    - precio: Precio unitario del ingrediente

public void reducirExistencias(int cantidad)

- **Descripción:** Reduce las existencias del ingrediente cuando se usa en un platillo.
- **Parámetros:**
    - cantidad: Cantidad a reducir del inventario
- **Excepciones:** Lanza IllegalArgumentException si la cantidad es mayor a las existencias

public boolean hayDisponibilidad(int cantidadRequerida)

- **Descripción:** Verifica si hay suficientes existencias para preparar un platillo.
- **Parámetros:**
    - cantidadRequerida: Cantidad necesaria del ingrediente

- **Retorna:** true si hay suficiente inventario, false en caso contrario

Clase: Platillo

Propósito: Representa un platillo del menú del restaurante.

Métodos Principales:

public Platillo(String nombre, ArrayList<Integer> ingredientes, double precioManoObra)

- **Descripción:** Constructor que crea un platillo con ID autogenerado.
- **Parámetros:**
    - nombre: Nombre del platillo
    - ingredientes: Lista de IDs de ingredientes necesarios
    - precioManoObra: Costo de mano de obra para preparar el platillo

public double calcularPrecioTotal(IngredienteDAO ingredienteDAO)

- **Descripción:** Calcula el precio total sumando mano de obra y costo de ingredientes.
- **Parámetros:**
    - ingredienteDAO: Objeto de acceso a datos para obtener precios de ingredientes
- **Retorna:** Precio total del platillo

public boolean verificarDisponibilidad(IngredienteDAO ingredienteDAO)

- **Descripción:** Verifica si todos los ingredientes están disponibles en inventario.
- **Parámetros:**
    - ingredienteDAO: Objeto de acceso a datos de ingredientes
- **Retorna:** true si todos los ingredientes están disponibles

Clase: Cliente

Propósito: Representa un cliente del restaurante.

Métodos Principales:

public Cliente(String dpi, String nombreCompleto, String usuario, String password, String rutaFoto)

- **Descripción:** Constructor que crea un cliente nuevo con tipo "normal".
- **Parámetros:**
    - dpi: Documento de identificación personal
    - nombreCompleto: Nombre completo del cliente
    - usuario: Nombre de usuario para login
    - password: Contraseña de acceso
    - rutaFoto: Ruta del archivo de imagen del cliente

public void incrementarPedidos()

- **Descripción:** Incrementa el contador de pedidos y actualiza el tipo de cliente si corresponde.
- **Lógica:** Si el cliente alcanza 10 pedidos pagados, cambia automáticamente a tipo "oro"

public boolean esClienteOro()

- **Descripción:** Verifica si el cliente tiene privilegios de cliente oro.
- **Retorna:** true si el cliente es tipo "oro"

Clase: OrdenTrabajo

Propósito: Representa una orden de trabajo para preparar un platillo.

Métodos Principales:

public OrdenTrabajo(Platillo platillo, Cliente cliente)

- **Descripción:** Constructor que crea una nueva orden con número correlativo.
- **Parámetros:**
    - platillo: Platillo a preparar
    - cliente: Cliente que solicita el platillo

public void cambiarEstado(EstadoOrden nuevoEstado)

- **Descripción:** Cambia el estado de la orden y registra la fecha/hora del cambio.

- **Parámetros:**
    - nuevoEstado: Nuevo estado (COLA_ESPERA, EN_COCINA, LISTO)

public void asignarCocinero(Cocinero cocinero)

- **Descripción:** Asigna un cocinero a la orden cuando pasa a preparación.
- **Parámetros:**
    - cocinero: Cocinero que preparará el platillo

Clase: Cocinero (Hilo)

Propósito: Representa un cocinero que prepara platillos de forma concurrente.

Métodos Principales:

public void run()

- **Descripción:** Método principal del hilo que procesa órdenes de trabajo.
- **Lógica:**
    - Toma órdenes de la cola de espera
    - Simula el tiempo de preparación
    - Mueve la orden a la lista de platillos listos

public void prepararPlatillo(OrdenTrabajo orden)

- **Descripción:** Simula la preparación de un platillo con tiempo de espera.
- **Parámetros:**
    - orden: Orden de trabajo a procesar
- **Funcionalidad:** Usa Thread.sleep() para simular tiempo de cocción

public boolean estaDisponible()

- **Descripción:** Verifica si el cocinero está disponible para tomar una nueva orden.
- **Retorna:** true si no está preparando ningún platillo

Clase: GestorColas

Propósito: Administra las colas de órdenes y la asignación de cocineros.

Métodos Principales:

public void agregarOrden(OrdenTrabajo orden)

- **Descripción:** Agrega una nueva orden al sistema y gestiona las colas.
- **Parámetros:**
    - orden: Nueva orden a procesar
- **Lógica:**
    - Si hay cocineros disponibles, asigna inmediatamente
    - Si no, coloca en cola de espera
    - Clientes oro tienen prioridad en la cola

public void procesarColaEspera()

- **Descripción:** Procesa la cola de espera cuando se libera un cocinero.
- **Lógica:** Toma la primera orden de la cola y la asigna a un cocinero disponible

public ArrayList<OrdenTrabajo> obtenerOrdenesEnEspera()

- **Descripción:** Retorna las órdenes que están esperando ser procesadas.
- **Retorna:** Lista de órdenes en cola de espera

Clase: IngredienteDAO

Propósito: Maneja la persistencia y operaciones CRUD de ingredientes.

Métodos Principales:

public void cargarDesdeArchivo(String rutaArchivo)

- **Descripción:** Carga ingredientes desde un archivo .igd.
- **Parámetros:**
    - rutaArchivo: Ruta del archivo de ingredientes
- **Formato:** id-nombre-marca-existencias-unidades-precio

public void guardarEnArchivoBinario(String rutaArchivo)

- **Descripción:** Serializa y guarda todos los ingredientes en archivo binario.
- **Parámetros:**

- rutaArchivo: Ruta donde guardar el archivo binario

public void agregarIngrediente(Ingrediente ingrediente)

- **Descripción:** Agrega un nuevo ingrediente al sistema.
- **Validaciones:** Verifica que no exista otro ingrediente con el mismo ID

Clase: AlgoritmoOrdenamiento

Propósito: Implementa algoritmos de ordenamiento para reportes.

Métodos Principales:

public static <T> void bubbleSort(ArrayList<T> lista, Comparator<T> comparador, VelocidadOrdenamiento velocidad)

- **Descripción:** Implementa el algoritmo Bubble Sort con velocidad configurable.
- **Parámetros:**
  - lista: Lista a ordenar
  - comparador: Criterio de comparación
  - velocidad: Velocidad de ordenamiento (ALTA, MEDIA, BAJA)

public static <T> void quickSort(ArrayList<T> lista, Comparator<T> comparador, VelocidadOrdenamiento velocidad)

- **Descripción:** Implementa el algoritmo Quick Sort con velocidad configurable.
- **Optimización:** Usa partición con pivote aleatorio para mejor rendimiento

public static long medirTiempoOrdenamiento(Runnable algoritmo)

- **Descripción:** Mide el tiempo de ejecución de un algoritmo de ordenamiento.
- **Retorna:** Tiempo en milisegundos

Clase: GeneradorReportes

Propósito: Genera reportes en PDF y gráficas.

Métodos Principales:

public void generarReporteClientes(String rutaArchivo)

- **Descripción:** Genera reporte de clientes con tabla y gráfica de pastel.

- **Contenido:**
    - Tabla con todos los datos de clientes
    - Gráfica de distribución por tipo (oro/normal)

public void generarTopIngredientesMasUsados(String rutaArchivo)

- **Descripción:** Genera reporte de top 10 ingredientes más usados.
- **Incluye:** Tabla de datos y gráfica de barras

public void generarTopPlatillosMasPedidos(String rutaArchivo)

- **Descripción:** Genera reporte de top 10 platillos más solicitados.
- **Requiere:** Datos históricos de órdenes de trabajo

Estructura de Archivos

Archivos de Datos

Ingredientes (.igd)

1-Salsa de tomate-Tomatodo-100-sobres-3.50

2-Queso cheddar-Quesos de Guatemala-12-sobres-10.00

Platillos (.pltll)

Pizza Hawaiana-1;2;3;4-15.00

Hamburguesa Clásica-2;5;6-12.50

Clientes (.clnt)

2345679890301-Tedy Lopez-ted-123-oro-C:\Users\Tedy\Desktop\foto.jpg

2345679890101-Roberto Lemus-ro-123-normal-C:\Users\Roberto\perfil.jpg

Archivos Binarios de Serialización

- ingredientes.dat: Datos serializados de ingredientes
- platillos.dat: Datos serializados de platillos
- clientes.dat: Datos serializados de clientes
- ordenes.dat: Historial de órdenes de trabajo

Consideraciones de Implementación

Concurrencia

- Los cocineros se implementan como hilos separados
- Se usa sincronización para evitar condiciones de carrera en las colas
- Los clientes oro tienen prioridad en la cola de espera

Persistencia

- Todos los datos se serializan en archivos binarios
- La aplicación restaura el estado anterior al iniciarse
- Se mantiene integridad referencial entre objetos

Validaciones

- Verificación de tipos de datos en formularios
- Validación de disponibilidad de ingredientes
- Control de acceso por tipo de usuario

Rendimiento

- Algoritmos de ordenamiento optimizados
- Gestión eficiente de memoria con ArrayLists
- Interfaz responsiva con hilos separados para tareas pesadas

Diagrama de flujo