**Aplicación de Clima (WeatherApp)**

* **Habilidades relevantes**: Java Swing, APIs RESTful (Open-Meteo), JSON parsing (json-simple), HTTPURLConnection, Manejo de excepciones, GUI Design, Programación orientada a objetos.
* **Descripción breve**: Aplicación de escritorio que muestra el clima actual de una ubicación específica, integrando datos de geolocalización y condiciones meteorológicas mediante APIs externas. Incluye una interfaz gráfica con imágenes dinámicas, temperatura, humedad, velocidad del viento y actualización en tiempo real.

**Componentes clave destacados:**

1. **Conexión a APIs**:
   * Uso de HttpURLConnection para solicitudes GET a las APIs de Open-Meteo (geolocalización y pronóstico del tiempo).
   * Procesamiento de respuestas JSON con la librería json-simple.
2. **Interfaz Gráfica (Swing)**:
   * Diseño de ventanas, campos de texto, botones e imágenes con JFrame, JLabel, JTextField, y JButton.
   * Carga de imágenes dinámicas según las condiciones climáticas (soleado, lluvia, nieve, etc.).
   * Manejo de eventos para la búsqueda en tiempo real con ActionListener.
3. **Lógica de Negocio**:
   * Conversión de códigos meteorológicos a descripciones legibles (ej: 0L → "Clear").
   * Cálculo del índice horario para filtrar datos relevantes del JSON.
   * Formateo de fechas y horas con LocalDateTime y DateTimeFormatter.
4. **Manejo de Errores**:
   * Validación de entradas del usuario (ej: texto vacío).
   * Verificación de códigos de respuesta HTTP (ej: 200 OK, 400 Bad Request).

**Nota adicional**: El proyecto demuestra integración de APIs externas, manipulación de datos JSON y creación de interfaces gráficas interactivas en Java, ideal para aplicaciones de escritorio multiplataforma. 🌦️

**Proyecto: Sistema ETL para Business Intelligence (HR Analytics)**

* **Habilidades relevantes**:  
  Python, Pandas (ETL), SQL (MySQL), SQLAlchemy (ORM), APIs de conexión (PowerBI), JSON/Excel parsing,  
  Gestión de credenciales, Programación orientada a objetos, Automatización de pipelines.
* **Descripción breve**:  
  Sistema que extrae datos de Excel, los transforma en JSON, los carga en una base de datos MySQL mediante transacciones seguras,  
  y permite su visualización en PowerBI. Diseñado para análisis de recursos humanos (ej: rotación de personal,  
  satisfacción laboral).

**Flujo del sistema:**

1. **Extract (Python)**:
   * JSONparser: Convierte hojas de Excel (ej: HR info.xlsx) en JSON estructurado con versionado por timestamp.
   * Limpieza inicial: Normaliza formatos (ej: Attrition → "Yes/No").
2. **Transform (Python + Pandas)**:
   * Validación de tipos de datos (numeric\_cols).
   * Conversión de columnas críticas (ej: OverTime a booleano).
   * Mapeo de valores nulos (pd.to\_numeric(errors='coerce')).
3. **Load (MySQL + SQLAlchemy)**:
   * Carga masiva con transacciones atómicas (engine.begin()).
   * Prevención de duplicados mediante ON DUPLICATE KEY UPDATE.
   * Conexión segura usando credenciales externas (credentialsDB).
4. **Visualización (PowerBI)**:
   * Conexión directa a MySQL para dashboards en tiempo real.
   * Métricas clave: Rotación de empleados, ingresos mensuales, años en la empresa, etc.

**Tecnologías clave integradas:**

| **Componente** | **Herramientas/Enfoque** |
| --- | --- |
| **Backend ETL** | Python, Pandas, JSON, SQLAlchemy, MySQL Connector |
| **Base de Datos** | MySQL (gestión de tablas empleados, consultas optimizadas) |
| **Visualización** | PowerBI (conexión ODBC/JDBC a MySQL, diseño de dashboards interactivos) |
| **Seguridad** | Credenciales externalizadas (archivo credentialsDB), prevención de inyección SQL |

**Puntos destacados:**

* **Automatización**:
  + Generación de nombres de archivo con timestamp (JSON\_2024-09-02\_14-30.json) para auditoría.
  + Input de usuario para rutas dinámicas de Excel (excelLocation = input(...)).
* **Buenas prácticas**:
  + Uso de transacciones para evitar corrupción de datos.
  + Validación de tipos de datos y limpieza antes de cargar en SQL.
  + Mensajes claros de éxito/error (✅/❌) para monitoreo.
* **Escalabilidad**:
  + Diseño modular: Separación entre conversión (JSONparser) y carga (JsonSQLUpload).
  + Preparado para integrar nuevas hojas de Excel o tablas SQL sin reescribir código.

**Relevancia profesional**:  
Este proyecto demuestra un flujo completo de **Data Engineering**, ideal para roles de BI Analyst, Data Engineer o backend developer. Combina manipulación de datos (Pandas), bases de datos relacionales (MySQL) y visualización (PowerBI), replicando escenarios reales de empresas. Su enfoque en seguridad y transacciones lo hace apto para entornos productivos. 📊🔗

### ****Proyecto: Sistema de Gestión de Recursos Informáticos (PC Manager)****

* **Habilidades relevantes**: Java (POO, Herencia, Polimorfismo), Diseño de jerarquías de clases, Métodos de instancia/static, Encapsulación, Manejo de paquetes.
* **Descripción breve**: Aplicación de consola que simula la gestión de recursos de hardware (laptops), software (instalación/compatibilidad) y usuarios (admin/invitado), con validaciones de memoria RAM, estado de antivirus, y autenticación de roles.

### ****Componentes clave****:

#### **1.** componentesPrincipales **(Clase Base)**

* **Funcionalidad**:
  + Define atributos base de hardware (hostName, CPU, RAM, ipAddress).
  + Método mostrarDetalles() para imprimir configuración en formato estructurado.
  + Punto de entrada (main) con flujo de demostración del sistema.
* **Técnicas destacadas**:
  + Uso de final para IP estática (inmutable).
  + Formateo de salida con printf.

#### **2.** Laptop **(Subclase de** componentesPrincipales**)**

* **Funcionalidad**:
  + Extiende la configuración base con atributos de pantalla (tipoDePantalla, tamanoDePantalla).
  + Sobrecarga de constructores para flexibilidad en la creación de objetos.
* **Técnicas destacadas**:
  + Herencia con extends.
  + Reutilización de métodos padre mediante super.mostrarDetalles().

#### **3.** Software **(Subclase de** Laptop**)**

* **Funcionalidad**:
  + Valida compatibilidad de software vs hardware (haySuficienteRam()).
  + Calcula memoria restante y determina si es ejecutable (esCompatible).
* **Técnicas destacadas**:
  + Operador ternario para mensajes condicionales ((esCompatible ? "Si" : "No")).
  + Lógica de negocio para gestión de recursos.

#### **4. Módulos Externos (Paquete** PC.Apps**)**

* **Componentes referenciados**:
  + installWiz: Gestión de instalación de aplicaciones.
  + AntiVirus: Detección de amenazas (ej: checkVirus()).
  + userManager: Sistema de autenticación con herencia (Admin/Guest).
* **Técnicas destacadas**:
  + Polimorfismo (userManager login = new Admin()).
  + Diseño modular para escalabilidad.

### ****Flujo de Ejecución (Demostración en**** main****)****

1. **Configuración de Hardware**:

java

Copy

Laptop laptop1 = new Laptop("Maria", "Intel I5", 16, "LED", 11);

Laptop laptop2 = new Laptop("Juan", "Intel I9", 64);

* + Crea laptops con configuraciones específicas (sobrecarga de constructores).

1. **Validación de Software**:

java

Copy

Software software1 = new Software(laptop1, "Teams", 1);

software1.haySuficienteRam(); // Output: "Hay memoria suficiente..."

* + Verifica si la RAM instalada soporta el software.

1. **Gestión de Aplicaciones**:

java

Copy

installWiz Teams = new installWiz("Teams");

Teams.checkApp(); // Verifica si está instalado

* + Simula instalación/consulta de apps.

1. **Seguridad y Usuarios**:

java

Copy

userManager login = new Admin();

login.login("admin","1234"); // Autenticación de administrador

* + Implementa control de acceso por roles.

### ****Buenas Prácticas Identificadas****:

| **Técnica** | **Ejemplo en Código** | **Beneficio** |
| --- | --- | --- |
| **Encapsulación** | private String hostName; + getters/setters | Protege el estado interno de los objetos |
| **Sobrecarga** | Dos constructores en Laptop | Flexibilidad al crear instancias |
| **Polimorfismo** | userManager login = new Admin() | Extensibilidad para nuevos roles |
| **DRY (Don't Repeat Yourself)** | Uso de super.mostrarDetalles() | Reduce redundancia de código |

**Relevancia Profesional**:  
Este proyecto ejemplifica el diseño de sistemas modulares en Java, aplicando principios SOLID (ej: Single Responsibility en clases separadas para hardware/software/usuarios). Es relevante para roles de desarrollo backend, ingeniería de software, o herramientas de administración de sistemas. 🖥️🔒

### ****Proyecto: Juego Piedra, Papel o Tijera (Web)****

* **Habilidades relevantes**: HTML5, CSS3 (Flexbox, Diseño Responsivo), JavaScript (DOM Manipulation, Event Handling), Lógica de Juegos.
* **Descripción breve**: Juego interactivo basado en navegador donde el usuario compite contra la computadora. Incluye interfaz minimalista, generación aleatoria de movimientos de la CPU, y sistema de reglas clásico con alertas de resultados.

### ****Componentes Clave****:

#### **1. Interfaz (HTML/CSS)**

* **Diseño Visual**:
  + Contenedor con bordes redondeados y paleta de colores tierra/verde (#81b622 como fondo principal).
  + Botones estilo "gaming" con hover implícito (efecto de borde #a47551).
  + Layout responsivo usando Flexbox (justify-content: space-evenly).
* **Estructura HTML**:
  + 3 botones con eventos onclick directos para capturar la elección del jugador.

#### **2. Lógica del Juego (JavaScript)**

* **Generación de Movimiento CPU**:

javascript

Copy

function pickComputerMove() {

const randomNumber = Math.random();

// Asignación equitativa (33.3% para cada opción)

}

* **Sistema de Reglas**:

javascript

Copy

if ((playerChoice === 'Rock' && computerMove === 'Paper') || ... ) {

result = "You loss"; // Condiciones de derrota

} else if (playerChoice === computerMove) { ... }

* **Feedback al Usuario**:
  + Alertas nativas con template strings para mostrar resultados:

javascript

Copy

alert(`You picked ${playerChoice}. Computer picked ${computerMove}. ${result}.`);

### ****Técnicas Destacadas****:

| **Área** | **Implementación** |
| --- | --- |
| **DOM Manipulation** | Uso de onclick en HTML para manejar interacciones. |
| **Randomización** | Math.random() con rangos definidos para elección de la CPU. |
| **Lógica Condicional** | Operadores lógicos (&&, ` |  | `) para cubrir todas las combinaciones posibles. |
| **Mantenibilidad** | Nombres claros en funciones/variables (playGame, pickComputerMove). |  |  |

### ****Flujo del Juego****:

1. **Elección del Jugador**: Click en botón (Rock/Paper/Scissors).
2. **Turno de la CPU**: pickComputerMove() genera una jugada aleatoria.
3. **Determinar Resultado**:
   * Piedra > Tijera
   * Papel > Piedra
   * Tijera > Papel
4. **Mostrar Resultado**: Alert con resumen de movimientos y resultado.

**Relevancia Profesional**:  
Este proyecto demuestra habilidades front-end esenciales: diseño web limpio, manipulación dinámica del DOM, y lógica de programación. Es escalable (ej: añadir puntuación o historial) y apto para portafolios de desarrollo web. 🌐🎮

### ****Proyecto: Automatizador de Búsqueda de Direcciones (Excel → Web)****

**Habilidades relevantes**: Python, Pandas (manipulación de datos), Tkinter (GUI), Web Scraping/APIs (webbrowser), Manejo de excepciones, Automatización de workflows.  
**Descripción breve**: Herramienta que automatiza búsquedas en Google Maps para direcciones almacenadas en Excel, con dos versiones:

* **CLI**: Interfaz de consola para procesamiento secuencial.
* **GUI**: Interfaz gráfica con seguimiento de índice, validación de entradas y feedback visual.

### ****Versiones del Proyecto****

#### **1. Versión CLI (Consola)**

* **Funcionalidad clave**:
  + Lee columnas DUNS\_STREET\_ADDR y DUNS\_PSTL\_CODE desde Excel.
  + Genera URLs de Google Maps concatenando dirección + código postal.
  + Abre automáticamente el navegador con la búsqueda (webbrowser.open()).
  + Permite navegar por índices con control manual (Y/N).
* **Técnicas destacadas**:
  + Uso de urllib.parse.quote\_plus() para formatear URLs válidas.
  + Validación de índices con manejo de errores (try-except, while loops).

#### **2. Versión GUI (Tkinter)**

* **Mejoras sobre la CLI**:
  + Interfaz visual con botones intuitivos y seguimiento de progreso.
  + Mensajes de error/éxito con messagebox.
  + Autoactualización del índice y diseño responsivo.
  + Paleta de colores profesional (#0c2d48, #2e8bc0).
* **Componentes Tkinter**:
  + Entry: Campo para ingresar índices.
  + askyesno(): Diálogo de confirmación para continuar.
  + Configuración estética: Botones con bordes personalizados, fuentes Roboto.

### ****Flujo de Datos (ETL Simplificado)****

1. **Extract**:

python

Copy

df = pd.read\_excel(file\_path, usecols=['DUNS\_STREET\_ADDR', 'DUNS\_PSTL\_CODE'], sheet\_name='Testing')

1. **Transform**:
   * Limpieza de ZIP codes (str(...)[:5]).
   * Concatenación: fullAddress = f"{dunsSTREET} {dunsZIP}".
2. **Load**:
   * Generación de URL: https://www.google.com/search?q={urllib.parse.quote\_plus(fullAddress)}.
   * Apertura automática en navegador.

### ****Buenas Prácticas Implementadas****

| **Área** | **Ejemplo** | **Impacto** |
| --- | --- | --- |
| **Manejo de Errores** | try-except alrededor de webbrowser.open() | Evita caídas por URLs malformadas |
| **Validación de Input** | Chequeo de 0 < index <= longitud | Previene índices inválidos |
| **UX** | Botón "Continuar" con autoincremento en GUI | Optimiza flujo de trabajo |
| **Seguridad** | Truncado de ZIP code a 5 dígitos | Protege datos sensibles |

### ****Posibles Mejoras (Roadmap)****

1. **Auto-Guardado**: Preservar el último índice usado entre sesiones.
2. **Logging**: Registrar búsquedas realizadas en un archivo .csv.
3. **Web Scraping**: Integrar BeautifulSoup para extraer coordenadas GPS directamente.
4. **Batch Processing**: Opción para procesar múltiples índices automáticamente.

**Relevancia Profesional**:  
Este proyecto es ideal para roles de **Data Analyst** o **Business Intelligence**, donde se requiere validar/geolocalizar datos masivos. Combina manipulación de datos (Pandas), automatización (webbrowser) y desarrollo de interfaces (Tkinter), replicando escenarios reales como limpieza de bases de direcciones o verificación de sucursales. 🗺️🔍

### ****Proyecto: Sistema de Autenticación (Login con Java Swing)****

* **Habilidades relevantes**: Java Swing (GUI), Event Handling, POO, Encapsulación, Validación Básica.
* **Descripción breve**: Interfaz gráfica de login con campos para usuario y contraseña, validación de credenciales estáticas ("123"/"123"), y botón para limpiar datos. Ideal como base para sistemas de acceso seguro.

### ****Componentes Clave****:

#### **1. Interfaz Gráfica (**testing **JFrame)**

* **Elementos UI**:
  + JTextField (Usuario) y JPasswordField (Contraseña).
  + Botones LoginBtn (Validar) y LoginLimpiarBtn (Limpiar).
  + Diseño minimalista con JPanel y ventana centrada (setLocationRelativeTo(null)).
* **Configuración Técnica**:
  + Tamaño fijo (225x235 px).
  + Cierre completo al salir (EXIT\_ON\_CLOSE).

#### **2. Lógica de Validación**

* **Autenticación**:

java

Copy

if (user.equals("123") && pass.equals("123")) {

JOptionPane.showMessageDialog(null, "Ingreso correctamente");

} else {

JOptionPane.showMessageDialog(null, "Contraseña incorrecta...");

}

* **Limpieza de Campos**:

java

Copy

public void limpiarDatos() {

LoginUserFld.setText("");

LoginPasswordFld.setText("");

}

#### **3. Manejo de Eventos**

* **ActionListeners**:
  + LoginBtn: Ejecuta validación y muestra mensajes con JOptionPane.
  + LoginLimpiarBtn: Llama a limpiarDatos() para resetear campos.

### ****Técnicas Destacadas****:

| **Área** | **Implementación** |
| --- | --- |
| **POO** | Clase testing heredando de JFrame para modularidad. |
| **Encapsulación** | Método limpiarDatos() reutilizado en múltiples acciones. |
| **Event-Driven** | Uso de ActionListener para manejo de clics. |
| **UX Básico** | Feedback visual mediante diálogos (JOptionPane). |

### ****Flujo de la Aplicación****:

1. **Inicio**:

java

Copy

SwingUtilities.invokeLater(() -> new testing()); // Lanza la GUI

1. **Interacción**:
   * Usuario ingresa credenciales → Click en "Login".
   * Validación estática → Mensaje de éxito/error.
2. **Salida**:
   * Cierre de ventana → Termina la aplicación.

### ****Posibles Mejoras (Escalabilidad)****:

1. **Conexión a Base de Datos**:
   * Validar credenciales contra una tabla SQL (ej: MySQL).
2. **Seguridad**:
   * Encriptación de contraseñas con BCrypt o SHA-256.
3. **Funcionalidad Adicional**:
   * Botón "Registrarse" para crear nuevos usuarios.
   * Recuperación de contraseña vía correo electrónico.
4. **Diseño Avanzado**:
   * Temas personalizados con FlatLaf o Material Design.

**Relevancia Profesional**:  
Este proyecto demuestra habilidades esenciales para desarrollar interfaces de usuario en Java, aplicables a sistemas empresariales (ej: software interno, herramientas de administración). Su estructura modular permite escalar hacia un sistema de autenticación robusto, relevante para roles de **Desktop Developer** o **Full-Stack Java Developer**. 🔒💻

### ****Proyecto: Sistema de Gestión Financiera (En Progreso)****

**Habilidades relevantes**: Java (POO, Herencia, Polimorfismo), Manejo de colecciones (HashMap), Diseño de jerarquías de clases.  
**Descripción breve**: Sistema para registrar transacciones financieras (ingresos/egresos), calcular impuestos diferenciados y generar reportes. Actualmente en fase de estructuración base.

### ****Componentes Clave Actuales****:

#### **1. Clase** Transaccion **(Abstracta)**

* **Funcionalidad**:
  + Base para transacciones con atributos: monto, descripción, fecha.
  + Métodos: procesarTransacción(), mostrarTransacciones(), calcularImpuestos().
  + Registro en HashMap (simula base de datos).
* **Detalles Técnicos**:
  + Constructor inicializa con "Capital Base" ($1500).
  + Impuesto genérico del 16% (calcularImpuestos()).

#### **2. Clase** Ingreso **(Hereda de** Transaccion**)**

* **Extensión**:
  + Sobreescribe calcularImpuestos() con tasa del 18%.

java

Copy

@Override

public void calcularImpuestos(double monto) { ... }

#### **3. Clase** Egreso **(Hereda de** Transaccion**)**

* **Estado Actual**:
  + Vacía (pendiente de implementación).
  + Posible uso: Registrar gastos con lógica específica (ej: categorización).

#### **4. Clase** Menu **(Punto de Entrada)**

* **Funcionalidad**:
  + Demuestra flujo básico: crear transacción → mostrar registros → calcular impuestos.
  + Usa HashMap<Integer, RegistroTransaccion> para almacenamiento temporal.

### ****Estructura Actual del Código****:

Copy

src/

├── menu/

│ └── Menu.java # Punto de entrada y "base de datos"

├── TransactionManager/

│ ├── Transaccion.java # Clase abstracta principal

│ └── TransaccionProcesos/

│ ├── Ingreso.java # Lógica de ingresos (impuesto 18%)

│ └── Egreso.java # Pendiente (ej: impuesto deducible)

### ****Análisis de Diseño****:

| **Aciertos** | **Oportunidades de Mejora** |
| --- | --- |
| ✅ Uso de herencia para diferenciar Ingreso/Egreso. | ❌ Transaccion no debería heredar de Menu (viola SRP\*). |
| ✅ Polimorfismo en calcularImpuestos(). | ❌ HashMap en Menu dificulta escalabilidad (debería estar en clase dedicada). |
| ✅ Encapsulación con clases internas (RegistroTransaccion). | ❌ Egreso sin implementar limita funcionalidad. |

\*SRP: Principio de Responsabilidad Única.

### ****Recomendaciones para Continuar****:

1. **Refactorizar Jerarquía**:
   * Mover baseDeDatos a clase GestorTransacciones (Single Responsibility).
   * Eliminar herencia Transaccion → Menu.
2. **Implementar Egreso**:

java

Copy

public class Egreso extends Transaccion {

@Override

public void calcularImpuestos(double monto) {

// Ej: Impuesto deducible del 10%

}

}

1. **Ampliar Funcionalidad**:
   * Añadir métodos para filtrar transacciones por tipo/fecha.
   * Integrar persistencia (ej: guardar en SQLite/archivo CSV).
2. **Mejorar Interacción**:
   * Menú interactivo (CLI) para crear/consultar transacciones.
   * Validación de montos negativos en Egreso.

**Relevancia Profesional**:  
Este proyecto sienta las bases para un sistema contable o de gestión fiscal. Dominar estas estructuras es clave para roles de **Backend Developer** o **Financial Software Engineer**, especialmente en sectores como fintech o ERP. 🧮💡

### ****Proyecto: Sistema de Gestión de Empleados (Employee Manager)****

**Habilidades relevantes**: Java (POO, Encapsulación), Swing (GUI básica), Manejo de arreglos, Algoritmos de búsqueda/cálculo.  
**Descripción breve**: Aplicación de escritorio para administrar empleados, con capacidad de registrar datos, calcular métricas salariales, buscar por nombre y generar reportes. Ideal para RRHH o gestión interna de equipos.

### ****Componentes Clave****:

#### **1. Clase** Empleado **(Modelo de Datos)**

* **Atributos**:  
  nombre, edad, departamento, salario (encapsulados con getters).
* **Método toString()**:  
  Formatea la información para mostrar en diálogos.

#### **2. Clase** GestionUsuario **(Lógica de Negocio)**

* **Funcionalidades**:
  + **Almacenamiento**: Arreglo estático de hasta 5 empleados (Empleado[] listaEmpleados).
  + **Operaciones**:
    - agregarEmpleado(): Añade empleados (con límite de capacidad).
    - salarioPromedio(): Calcula el promedio salarial.
    - menorSalario()/mayorSalario(): Identifica salarios mínimo/máximo.
    - nameSearch(): Búsqueda por nombre.
    - showAll(): Reporte completo de empleados.

#### **3. Clase** EmployeeManager **(Interfaz de Usuario)**

* **Menú Interactivo**:  
  Opciones accesibles via JOptionPane (Swing):

java

Copy

JOptionPane.showInputDialog("1) Agregar Empleado\n2) Mostrar Salario Promedio...");

### ****Análisis Técnico****:

| **Aciertos** | **Oportunidades de Mejora** |
| --- | --- |
| ✅ Encapsulación correcta en Empleado. | ❌ **Almacenamiento**: Arreglo estático limita a 5 empleados (usar ArrayList). |
| ✅ Métodos de búsqueda eficientes (O(n)). | ❌ **Redundancia**: Métodos agregarEdad(), agregarDepartamento() no son funcionales. |
| ✅ Feedback visual con Swing. | ❌ **Validación**: No verifica valores negativos en edad/salario. |

### ****Recomendaciones para Mejorar****:

1. **Estructura de Datos Dinámica**:

java

Copy

private List<Empleado> listaEmpleados = new ArrayList<>();

1. **Validación de Inputs**:

java

Copy

// En el main(), al ingresar salario:

if (salario < 0) throw new IllegalArgumentException("Salario no válido");

1. **Eliminar Métodos Redundantes**:  
   Borrar agregarEdad(), agregarDepartamento(), agregarSalario().
2. **Manejo de Errores**:

java

Copy

try {

edad = Integer.parseInt(JOptionPane.showInputDialog(...));

} catch (NumberFormatException e) {

JOptionPane.showMessageDialog(null, "Edad debe ser un número");

}

1. **Mejorar Reportes**:  
   Usar StringBuilder en showAll() para eficiencia con muchos registros.

### ****Flujo de la Aplicación****:

1. **Agregar Empleado**:
   * Inputs via diálogos → Crear objeto Empleado → Almacenar en listaEmpleados.
2. **Métricas Salariales**:
   * **Promedio**: Suma salarios / cantidad empleados.
   * **Extremos**: Iteración comparativa (Double.MAX\_VALUE y Double.MIN\_VALUE).
3. **Búsqueda por Nombre**:
   * Comparación lineal (equals()) → Retorna Empleado o null.

### ****Ejemplo de Uso****:

java

Copy

// Agregar empleado:

Empleado juan = new Empleado("Juan", 30, "IT", 5000.0);

miGestion.agregarEmpleado(juan);

// Buscar salario más alto:

Empleado top = miGestion.mayorSalario();

System.out.println(top); // "El empleado Juan... salario de 5000.0"

**Relevancia Profesional**:  
Este proyecto es relevante para roles de **Backend Developer** o **Desktop App Developer**, demostrando manejo de estructuras de datos, algoritmos básicos e interfaces gráficas simples. Perfecto para portafolios enfocados en aplicaciones empresariales. 👨💼📊

### ****Proyecto: Calculadora Básica (En Progreso)****

**Habilidades relevantes**: Java Swing (GUI), Manejo de Eventos, POO.  
**Descripción breve**: Interfaz gráfica de una calculadora con funcionalidad básica para sumar, restar y multiplicar. Actualmente en fase inicial de diseño de UI y captura de inputs.

### ****Estado Actual****:

#### **1. Interfaz Gráfica (CalculadoraUI)**

* **Componentes Implementados**:
  + Botones numéricos (0-9).
  + Botones de operaciones: +, -, \*.
  + Campo de texto (Output) para mostrar la expresión.
  + Botones Borrar y Total (sin funcionalidad aún).
* **Diseño**:
  + Ventana de 295x258 píxeles, centrada en pantalla.
  + Layout organizado con JPanel (posiblemente usando un GridLayout no mostrado en el código).

#### **2. Funcionalidad Parcial**:

* **Input de Números/Operadores**:

java

Copy

// Ejemplo para botón "1":

CalculadoraBTN1.addActionListener(e -> Output.setText(Output.getText() + " " + 1));

* **Limitaciones**:
  + No hay lógica para calcular resultados (=) o borrar (C).
  + Las operaciones se concatenan con espacios (ej: 3 + 5), pero sin evaluación.

### ****Análisis de Código****:

| **Aciertos** | **Pendientes/Oportunidades** |
| --- | --- |
| ✅ Estructura base de UI con Swing. | ❌ **Falta lógica de cálculo**: Evaluar expresiones como 3 + 5 \* 2. |
| ✅ Manejo básico de eventos para inputs. | ❌ **Botones inactivos**: CalculadoraTotalBtn y CalculadoraBorrarBtn no tienen ActionListener. |
| ✅ Separación clara entre UI y lógica de interacción. | ❌ **Sin validación de errores**: Ej: división por cero, entradas inválidas. |

### ****Recomendaciones para Continuar****:

1. **Implementar Lógica de Cálculo**:
   * Crear clase CalculadoraLogica con métodos para evaluar expresiones (ej: usando ScriptEngine o parser personalizado).

java

Copy

public class CalculadoraLogica {

public static double evaluar(String expresion) {

// Ejemplo usando ScriptEngine (para soportar +, -, \*, /):

ScriptEngineManager mgr = new ScriptEngineManager();

ScriptEngine engine = mgr.getEngineByName("JavaScript");

return (double) engine.eval(expresion.replace(" ", ""));

}

}

1. **Agregar Funcionalidad a Botones Pendientes**:
   * **Botón "=" (CalculadoraTotalBtn)**:

java

Copy

CalculadoraTotalBtn.addActionListener(e -> {

String expresion = Output.getText().replace(" ", "");

double resultado = CalculadoraLogica.evaluar(expresion);

Output.setText(String.valueOf(resultado));

});

* + **Botón "C" (CalculadoraBorrarBtn)**:

java

Copy

CalculadoraBorrarBtn.addActionListener(e -> Output.setText(""));

1. **Añadir División y Decimales**:
   * Botón / y . para operaciones completas.

java

Copy

// Ejemplo para botón ".":

JButton CalculadoraDecimalBtn = new JButton(".");

CalculadoraDecimalBtn.addActionListener(e -> Output.setText(Output.getText() + "."));

1. **Mejorar UX/UI**:
   * Usar GridLayout para alinear botones simétricamente.
   * Añadir colores y fuentes consistentes.
2. **Manejar Errores**:

java

Copy

try {

double resultado = CalculadoraLogica.evaluar(expresion);

} catch (Exception ex) {

JOptionPane.showMessageDialog(null, "Error en la expresión");

}

### ****Ejemplo de Flujo de Trabajo****:

1. **Usuario Ingresa**: 3 + 5 \* 2 → Output muestra 3 + 5 \* 2.
2. **Click en "="** → Lógica evalúa 3+5\*2 = 13 → Output actualizado a 13.
3. **Click en "C"** → Output se borra.

**Relevancia Profesional**:  
Este proyecto es fundamental para demostrar competencias en desarrollo de interfaces gráficas y lógica de aplicaciones, relevante para roles como **Desktop Developer** o **Java Swing Developer**. Al completarse, sería un ejemplo sólido de aplicación funcional con POO y manejo de eventos. 🔢🖥️

# Análisis Integral del Sistema de Gestión Bancaria

## Habilidades Clave

* **Java, POO, Swing (GUI)**
* **Estructuras de Datos** (Colas, Pilas, Listas Enlazadas, Mapas, PriorityQueue)
* **Persistencia de Datos** (Archivos TXT, Manejo de Transacciones)
* **Web Scraping** (Jsoup para tipo de cambio)
* **Manejo de Tiempo** (LocalDateTime, Duration)

## Descripción General

Sistema modular para gestionar colas de atención en un banco, con configuración personalizable, tiquetes de clientes, manejo de cajas (preferenciales, rápidas, regulares), reportes estadísticos y consulta en tiempo real del tipo de cambio.

## **Componentes Clave y Problemas Detectados**

### 1. ****Configuración del Banco (ConfiguracionBanco)****

* Define nombre del banco y cantidad de cajas (mínimo 3).
* Guarda/recupera configuración en configuracion.txt.
* **Problema:** Inconsistencia en nombres de archivos (crearTxt() usa prod.txt, pero cargarConfiguracion() busca configuracion.txt).
* **Recomendación:** Unificar nombres de archivos en todos los métodos.

### 2. ****Gestión de Colas y Tiquetes (GestionTiquetes, ColaClientes)****

* **Estructuras utilizadas:**
  + Lista circular doblemente enlazada para priorizar tiquetes.
  + **Colas FIFO** para cajas preferenciales/rápidas.
  + **Mapeo dinámico de colas (HashMap<Integer, ColaClientes>)** para balanceo de carga en cajas regulares.
* **Problemas:**
  + **Complejidad innecesaria:** La lista circular podría reemplazarse con PriorityQueue.
  + **Lógica confusa:** imprimirEstadoCola() llama a atenderTiquete(), mezclando visualización con acción.
* **Recomendaciones:**
  + Reemplazar lista circular por PriorityQueue.
  + Introducir interfaz IGestionTiquetes para desacoplar GestionTiquetes de Caja.

### 3. ****Registro y Reportes (Registro, transacciones.txt)****

* **Métricas Clave:**
  + Tiempo promedio de atención por caja.
  + Caja con mayor número de clientes atendidos.
  + Tiempo total de atención.
* **Problema:** transacciones.txt se lee completamente en cada reporte, lo que afecta el rendimiento.
* **Solución:** Usar caché en memoria con actualización periódica.

### 4. ****Web Scraping (WebScraper - Consulta Tipo de Cambio)****

* **Problema:** Si Google cambia su estructura HTML, el scraper falla.
* **Solución:** Manejar excepciones con try-catch y mostrar mensajes claros al usuario.
* **Alternativa:** Reemplazar scraping por API como OpenExchangeRates.

### 5. ****Seguridad en Archivos****

* **Problema:** transacciones.txt no tiene backup ante corrupción.
* **Solución:** Implementar sistema de versionado o logs binarios.

## **Diagrama de Flujo del Sistema**

flowchart TD

A[Usuario: Menú Principal] --> B[Agregar Cliente]

B --> C{¿Tipo de Cliente?}

C -->|Preferencial| D[Encolar en Caja Preferencial]

C -->|Urgente| E[Encolar en Caja Rápida]

C -->|Regular| F[Asignar a Caja Menos Ocupada]

A --> G[Atender Cliente]

G --> H{¿Caja Válida?}

H -->|Sí| I[Atender y Registrar]

A --> J[Generar Reporte]

J --> K[Mostrar Métricas]

A --> L[Consultar Dólar]

## **Mejoras de Código Críticas**

### ****1. Corregir Persistencia en Configuracion****

// En crearTxt():

FileWriter fileWriter = new FileWriter("configuracion.txt");

### ****2. Implementar toString() en NodoC****

public String toString() {

return "Tiquete ID: " + tiquete.getIdCliente() + ", Descripción: " + tiquete.getDescripcion();

}

### ****3. Optimizar Reportes con Caché en Memoria****

private Map<Integer, Reporte> cacheReportes = new HashMap<>();

### ****4. Validación de Inputs****

char tipo = // ...input...

if (tipo != 'P' && tipo != 'A' && tipo != 'R') {

JOptionPane.showMessageDialog(null, "Tipo no válido. Use P, A o R.");

continue;

}

### ****5. Manejo de Errores en Web Scraping****

// En WebScraper.cambioDolar():

catch (IOException e) {

JOptionPane.showMessageDialog(null, "Error al obtener el tipo de cambio.");

}

## **Propuestas para Versión 2.0**

* **Interfaz Gráfica Avanzada:** Usar JavaFX o Swing con tablas en tiempo real.
* **API para Tipo de Cambio:** Reemplazar scraping por OpenExchangeRates.
* **Priorización Dinámica:** Implementar colas prioritarias para clientes VIP.
* **Auditoría de Transacciones:** Firma digital en transacciones.txt.
* **Simulación en Tiempo Real:** Mostrar animación de colas con hilos (Thread).

## **Conclusión**

Este proyecto demuestra competencias sólidas en desarrollo de software empresarial, con potencial para convertirse en una herramienta profesional de gestión bancaria. Con las mejoras propuestas, alcanzaría un nivel de producción optimizado. 🏦🚀

New chat