Práctica: Caja de Cafetería Universitaria

[ITI | Programación 1 | UTN | Java | GUI | BD]

Objetivo general

Desarrollar una aplicación de escritorio en Java con interfaz gráfica *(Swing/JavaFX)*, orientada a objetos *(POO/OOP)* y modular, conectada a una base de datos local para gestionar ventas de una cafetería universitaria. El sistema debe incluir inicio de sesión *(Encriptado con hash SHA-256)*, menús, operaciones matemáticas, control de flujo *(if/else, switch),* bucles/ciclos *(for, while-do, do-while)*, manejo de errores con *try-catch*, y registro de sucesos y errores en la base de datos. Construcción con Maven o Ant y versionado en GitHub.

Contexto funcional

El cajero debe poder:

1. Iniciar sesión con usuario y contraseña *(encriptada en hash SHA-256)* almacenados en la Base de Datos.
2. Gestionar productos: CRUD - crear, ver/listar, editar/actualizar, y en vez de borrar se debería poder activar/inactivar (desactivar).
3. Registrar una venta: agregar ítems consecutivos con bucle hasta finalizar, calcular subtotal, impuesto (IVA del 7% e IVI del 13%), descuento (si lo desea agregar) y total; persistir todo en la Base de Datos (factura con cabecera y detalle).
4. Consultar ventas del día, y productos vendidos.
5. Usar una calculadora con operaciones básicas y dos botones adicionales: *potencia (elevación) y porcentaje*.
6. Imprimir (descargar) un ticket (bill, recibo, factura) simple (TXT/PDF) de ventas.

Requisitos técnicos mínimos

1) OOP y modularidad

Estructura de paquetes sugerida:

```
app             # arranque (main)
ui              # ventanas y diálogos
dominio         # entidades (Usuario, Producto, Venta, DetalleVenta)
servicio        # lógica (AuthService, VentaService, ProductoService)
infrastructura  # persistencia (Bases de Datos, Repositorios, Logs)
utilidades      # validaciones, formatos
```

Patrón sugerido: Arquitectura en capas.

2) Interfaz gráfica *(Swing)*

- Ventana de Log-in que valida contra la Bases de Datos.
- *Ventana Principal con barra de menús:* Archivo, Productos, Ventas, Herramientas (Calculadora), Ayuda.

- Validaciones de campos y mensajes de usuario con JOptionPane.

3) Base de datos

Tablas mínimas (ajuste de tipos según motor):

```
USUARIOS(id PK, username UNIQUE, password hashed, rol, activo, creado DateTime)

PRODUCTOS(id PK, nombre completo, precio_unitario, activo, creado DateTime)

VENTAS(id PK, user_id FK, fecha_hora DateTime, subtotal, impuestoIVA,
impuestoIVI, descuento, total)

DETALLES_VENTA(id PK, venta_id FK, product_id FK, cantidad, precio_unit,
total_linea)

LOGS(id PK, fecha_hora, nivel, evento, detalle, stacktrace -del try-catch-)
```

Login: comparar username y password (encriptado con hash SHA-256). Usar *PreparedStatements* para evitar inyección SQL. Insertar en LOGS eventos clave y cualquier error con stacktrace.

4) Control de flujo, ciclos y errores

- Usar if/else, switch y al menos tres bucles (for, while-do, do-while) en el flujo.
- Manejo de excepciones con try-catch, mensajes amigables y registro en LOGS.

5) Operaciones matemáticas

- **Venta:** subtotal, impuestos, descuento y total.
- **Calculadora:** +, −, ×, ÷ (validar ÷ 0), MOD y dos operaciones extra: *potencia (elevación) y porcentaje*.

6) Repositorio GitHub

- Repositorio con mínimo 12 commits por estudiante o 6 por integrante (si son parejas) o 4 c/u si tríos.
- Mensajes de commit significativos: características (feat), arreglo (fix), y refactor (refactorización).
- README.md con requisitos, compilación, ejecución y scripts SQL.

7) UML del Diagrama de Clases POO/OOP

- Diagrama de clases del modelo de dominio.

Librerías y API's recomendadas

- **JDK estándar:** javax.swing, java.sql, java.util.logging, java.security.MessageDigest.
- **Logging:** java.util.logging.
- **JDBC**: driver del motor elegido para librería de la BD (JAR local o dependencia de Maven).
- JUnit básico para pruebas unitarias de código.

FrameWorks/API's que pueden investigar: Spring, Hibernate/JPA, API REST/HTTP/JSON externos, Swagger.

Entregables

1) Código fuente del proyecto en ZIP o RAR.
2) README.md, y el link (hipervínculo) de GitHub.
3) Archivo JAR.

4) Scripts SQL de creación de BD y datos llenos.
5) Diagrama UML de Clases POO/OOP.
6) Evidencia (*capturas/screenshots:* log-in, productos, venta, calculadora, historial, datos encriptados).
7) Back-up de la BD llena.

Criterios de evaluación (100 pts)

- OOP y modularidad (MVC/capas) – 20 pts
- GUI Swing – 15 pts
- Conexión BD + JDBC + PreparedStatements – 20 pts
- Login con hash + control de intentos – 8 pts
- Flujo de venta (bucles, IVA, descuento) – 10 pts
- Manejo de errores try-catch + UI – 7 pts
- Logs en BD (sucesos + errores) – 10 pts
- Calculadora (6 operaciones) – 5 pts
- Build (Ant/Maven) – 3 pts
- GitHub – 2 pts

Reglas académicas

- Trabajo individual o en parejas o máximo tríos; se exige originalidad, sin IA, y sin copia entre grupos.

Pistas y sugerencias

- Encriptación por Hash de Contraseña (SHA-256) con *MessageDigest.getInstance("SHA-256")*.
- ***Errores típicos:*** conexión fallida, credenciales inválidas, formato numérico, ÷0, violaciones FK/UK y PK/FK.
- ***Modularidad:*** Separación UI/negocio/persistencia; la UI no debe llamar JDBC directamente (ciberseguridad).