Rúbrica de Evaluación Sumativa Nº3 explicada

⭐ ¿Qué debo tener para obtener 100% en todo?

1. **GUI completa** con campos, botones y tabla funcionando ✓
2. **CRUD totalmente funcional** conectado a SQLite ✓
3. **MVC bien aplicado** sin mezclar capas ✓
4. **Proyecto modular** con paquetes organizados ✓
5. **Buenas prácticas** en nombres, estructura, y estilo de código ✓
6. **Aplicación funcionando sin errores de compilación ni ejecución** ✓

✅ IL 3.1 – Construcción de la interfaz gráfica (30%)

Rúbrica (Muy buen desempeño – 100%)

> *"Construye la interfaz gráfica con TODOS los controles y componentes para capturar TODOS los datos y muestra los reportes de información con el fin de cumplir con todos los requisitos del caso."*

¿Qué deben tener implementado para lograr el 100%?

✔ Una interfaz hecha en Swing completa y funcional
✔ Todos los campos necesarios según el caso (ej.: id, nombre, precio, stock)
✔ Botones operativos: Nuevo, Guardar, Actualizar, Eliminar, Buscar
✔ Una tabla (`JTable`) que muestre correctamente los datos (listar, refrescar)
✔ Formularios organizados con contenedores adecuados (paneles, layouts)
✔ Validaciones mínimas (no dejar campos obligatorios vacíos)
✔ La aplicación debe **compilar y correr sin errores**

En resumen: la GUI debe permitir ingresar datos, mostrarlos y operar el CRUD de forma clara y completa.

✅ IL 3.2 – Aplicación del patrón MVC (20%)

Rúbrica (Muy buen desempeño – 100%)

> *"Separa COMPLETAMENTE en capas de responsabilidad cada uno de los elementos que forman el proyecto, implementando la integración entre capas."*

¿Qué deben tener implementado para lograr el 100%?

✔ Separación clara por paquetes:

- `model` → clases de datos (POJOs)
- `dao` → consultas y conexión con SQLite
- `controller` → interacción vista ↔ lógica
- `view` → formularios Swing
- `util` → conexión, helpers

✔ La vista no tiene lógica de negocio (solo interfaz)
✔ El controlador coordina operaciones llamadas desde la vista
✔ El DAO gestiona toda la comunicación con SQLite
✔ El modelo representa las entidades de la base de datos
✔ Todo está **bien conectado** entre sí y funciona

✅ IL 3.3 – Programación de consultas (CRUD) (30%)

Rúbrica (Muy buen desempeño – 100%)

> *"Codifica correctamente todos los métodos (insertar, modificar, eliminar, buscar y listar elementos) para gestionar la información."*

¿Qué deben tener implementado para lograr el 100%?

✔ Todos los métodos CRUD funcionando correctamente:

- `insertar()`
- `modificar()`
- `eliminar()`
- `buscarPorId()`
- `listar()`

✔ Uso adecuado de `PreparedStatement`
✔ Manejo de excepciones (`SQLException`)
✔ Integración del CRUD con la interfaz:

- Guardar agrega nuevos registros
- Actualizar modifica el registro seleccionado
- Eliminar borra correctamente
- Buscar rellena los campos en la interfaz
- Listar actualiza la tabla visual

✔ Los datos quedan realmente guardados en el archivo SQLite (`productos.db`)

✅ IL 3.4 – Modularidad y agrupación correcta del código (10%)

Rúbrica (Muy buen desempeño – 100%)

> *"Separa en capas de responsabilidad cada uno de los elementos que forman el proyecto, implementando de manera correcta la integración entre ellas."*

¿Qué deben tener implementado para lograr el 100%?

✔ Código organizado en módulos/paquetes coherentes:

- `view`
- `model`
- `controller`
- `dao`
- `util`

✔ Ninguna clase duplicada, sin código disperso
✔ Métodos correctamente ubicados:

- La vista llama al controlador
- El controlador llama al DAO
- El DAO llama a la BD
  ✔ Proyecto ordenado, fácil de navegar

✅ IL 3.5 – Estándares y buenas prácticas de desarrollo (10%)

Rúbrica (Muy buen desempeño – 100%)

> *"Codifica siguiendo convenciones para nombres de paquetes, clases, variables/constantes y métodos. Además, separa los componentes del proyecto en diferentes paquetes de acuerdo con su función."*

¿Qué deben tener implementado para lograr el 100%?

✔ Nombres correctos según estándares Java:

- Paquetes en minúsculas → `cl.duoc.tienda.dao`
- Clases en PascalCase → `ProductoDAO`, `FrmProductos`
- Métodos en camelCase → `insertarProducto()`, `cargarTabla()`
- Variables en camelCase → `txtNombre`, `productosList`

✔ Código limpio:

- Sangría correcta
- Comentarios útiles (no excesivos)
- Código legible (sin líneas repetidas ni basura)

✔ Proyecto organizado y profesional
✔ No hay lógica dentro de la vista que debería ir en el controlador o DAO

En resumen: el estudiante demuestra que domina buenas prácticas y escribe código mantenible.