Práctica de Programación 1

Java (NetBeans) + Swing + JDBC + (SQL Server | PostgreSQL | Oracle 21c XE/23ai FREE)

Modalidad: Individual

Objetivo General

Diseñar y construir, desde cero, una aplicación de escritorio que gestione tareas, aplicando Programación Orientada a Objetos (POO), GUI basada en eventos con Swing de Ambiente Gráfico de Java, manejo de excepciones, estructuras de datos dinámicas y persistencia con JDBC sobre una base de datos permitida. Versionar el trabajo con Git y publicar en un remoto (GitHub o Bitbucket), sin utilizar entornos de desarrollo colaborativo, ni asistentes de IA.

Reglas y Alcances *(Obligatorias)*

- **Lenguaje/IDE:** Java en NetBeans *(Ant o Maven a elección)*.
- **Interfaz Gráfica:** Swing o JavaFX *(JFrame/JPanel + controles estándar y eventos)*.
- **Base de Datos:** *Elija solo una* — SQL Server, PostgreSQL u Oracle *(21c XE o 23ai FREE)*.
- **Persistencia:** JDBC/OBDC con uso de *Prepared Statements* y transacciones *(cuándo corresponda)*.
- **Arquitectura:** Paquetes separados *(Dominio, DAO, Servicio, UI-UX, App)* respetando la modularidad.
- **Estructuras Dinámicas:** Utilice al menos dos *(Arreglos, Diccionarios, Listas, Listas de Arreglos, Mapas, Mapas de Hash, Array Deque, Grafos, Árboles, Matrices, Cubos, Colecciones, u otras estructuras dinámicas)*.
- **Excepciones:** Implemente try–catch, al menos una excepción personalizada y estrategias de recuperación *(la app no debe cerrarse por errores controlables)*.
- **Paradigmas:** Incluya una demostración breve de lenguaje estructurado, POO/OOP y paradigma declarativo/funcional *(Streams)* en una sección evidenciable.
- **Control de Versiones:** Git con commits frecuentes, varios, muchos y con significado; publicar en GitHub o Bitbucket.
- **Propiedad Intelectual:** el código debe ser 100% original; no se permite uso de asistentes de IA para generar código, ni de "apoyarse" (copiar, usar, reutilizar, robar) el código de algún compañero/a.

Requerimientos Funcionales Mínimos

1. **Crear tarea con:** título, prioridad *(1=Alta, 2=Media, 3=Baja)*, fecha opcional y marca especial *(★)*.
2. Listar tareas en una tabla con orden lógico *(por ID o por fecha)*.
3. **Alternar el Estado:** Hecho/Pendiente.
4. Eliminar tarea *(desasignar, no mostrar, pero no borrar de base de datos)*.
5. Deshacer la última eliminación mediante una estructura tipo pila/cola *(límite razonable)*.
6. **Persistencia:** los cambios deben permanecer tras cerrar y reabrir la aplicación.
7. **Validación:** impedir títulos vacíos, prioridades fuera de rango u otros datos inválidos.
8. **Mensajería:** mostrar mensajes de error comprensibles al usuario y mantener registro mínimo *(consola/archivo)*.

Requerimientos No Funcionales

- **POO/OOP:** buena abstracción, encapsulamiento, herencia *(al menos una sub-clase)* y polimorfismo *(inclusión y/o sobrecarga)*. Realizar diagrama.
- **Modularidad:** separación clara por capas *(UI/UX, Servicio/Negocio, Acceso a Datos/DAO, Dominio)*.
- **Transacciones:** operaciones multi-paso deben ser atómicas *(completan o se revierten)*.
- **Pruebas manuales:** elaborar y documentar un check-list propio.
- **UML:** diagrama de clases simple de Bases de Datos y de POO/OOP *(imágen, PDF, en Word, o escaneado)*.

Base de Datos

Defina el modelo de datos mínimo para la entidad principal "Tarea" *(ID, título, prioridad, estado, especial, fecha y campos de auditoría)*. Establezca clave primaria, restricciones e índices razonables. Normalice. Agregue el controlador JDBC correspondiente al proyecto y documente en el README.md: proveedor elegido, versión del servidor/driver, formato de URL JDBC, esquema/usuario, plantilla de la clase de conexión y proceso de inicialización.

La aplicación debe ser capaz de crear la tabla/esquema, si no existe, o incluir un procedimiento de inicialización propio y, documentado. No se incluyen fragmentos de SQL en esta guía; el estudiante debe escribirlos.

Interfaz Gráfica *(Mínima)*

- Ventana principal con formulario *(título, prioridad, fecha, especial)* y botones: *Agregar, Alternar Hecho, Eliminar, Deshacer*.
- Tabla con columnas adecuadas *(ID, Título, Prioridad, Estado, etc...)* y selección de filas.
- Eventos bien gestionados *(listeners)*, sin bloquear la UI/GUI.
- Principios de diseño: claridad, consistencia y retroalimentación *(por ejemplo: barra de estado)*.

Cronograma Sugerido

- **H1 (60 min):** Normalización de la Base de Datos. Demostración breve de paradigmas (imperativo vs POO vs declarativo con Streams) y diseño general con diagramas UML correspondientes.
- **H2 (60 min):** Implementación del dominio *(entidades, validaciones, excepción personalizada)* y selección de colecciones. Versionado con Git.
- **H3 (60 min):** Construcción de la GUI *(formulario + tabla + eventos)* y flujo en memoria. Versionado con Git.
- **H4 (60 min):** Integración con JDBC y la BD elegida (driver, conexión, creación de esquema, CRUD, transacciones, ReadMe.md). Versionado con Git.
- **H5 (60 min):** Pruebas manuales, ajustes finales, capturas; versionado con Git y publicación en remoto.

Checklist de Aceptación

[ ] La aplicación abre sin errores y muestra la GUI.
[ ] Crear tarea válida la inserta en tabla y BD; al reiniciar, persiste.
[ ] Alternar estado refleja el cambio en la tabla y en la BD.
[ ] Eliminar no remueve de tabla y BD, pero nada se muestra.
[ ] Deshacer restaura la última eliminación *(límite razonable)*.
[ ] Validaciones impiden datos inválidos con mensajes claros y sin cierre de la aplicación.
[ ] Se utilizaron al menos dos colecciones dinámicas en la lógica.
[ ] Normalización, UML DB y POO/OOP, y README completos.
[ ] Git remoto con historial de commits significativos.
[ ] Documentación, Empaquetado, Sistema modular, Uso de Control de Flujos y Excepciones.

Entregables

- Repositorio remoto (GitHub o Bitbucket) con el proyecto NetBeans.
- **README.md con:** BD elegida, versión/driver, formato de URL JDBC *(descrito por usted)* con plantilla/template de clase de conexión, y pasos para ejecutar.
- Sección de paradigmas con comparación breve *(5–10 líneas cada uno)*.
- Decisiones de diseño *(capas, colecciones usadas y justificación)*.
- Capturas de la GUI y evidencia de persistencia tras reinicio.
- Notas sobre transacciones y manejo de excepciones.
- Diagrama UML *(imagen, escaneado, Word o PDF)*.
- Checklist marcado por el estudiante y documento completo con requeridos.

Rúbrica de Evaluación

Criterio	Puntos
Paradigmas explicados con ejemplos propios y documentación general	**10**
POO correcta *(abstracción, encapsulamiento, herencia, polimorfismo, modularidad, diagrama UML)*	**25**
GUI Swing + Eventos *(usabilidad básica, CRUD a BD y estabilidad)*	**25**
Manejo de Excepciones + Recuperación *(incluye excepción personalizada)*	**15**
Estructuras dinámicas *(uso pertinente y justificado)*, ciclos, bucles, control de flujo	**10**
Persistencia JDBC real *(CRUD desde BD + transacciones + arranque limpio)*	**10**
Git remoto y commits significativos	**5**

Recomendaciones para el Estudiante

- Antes de programar, defina el modelo de datos y reglas de validación.
- Documente el formato de fecha a usar y mantenga consistencia GUI ↔ BD.
- Aísle el acceso a datos en una capa DAO; la UI no debe hablar directo con JDBC.
- Pruebe fallas controladas *(Por ejemplo: Prioridad fuera de rango, nulos, errores, datos erróneos, et…)* y verifique la recuperación *(deben seguir los datos correctos, aun si cierro el programa y luego lo vuelvo a correr)*.
- Haga commits pequeños y frecuentes con mensajes claros.
- Incluya una breve sección de lecciones aprendidas al finalizar la documentación.