Universidad Técnica Nacional

Sede Regional San Carlos

Ingeniería en Tecnologías de Información

Logotipo

Descripción generada automáticamente

Programación I, II-25, SC-01

Práctica sistema de notas

Profesor:

[Andrés Joseph Jiménez Leandro](https://campusvirtual.utn.ac.cr/user/view.php?id=18905&course=84030)

Propuesto por:

Jairo Steven Herrera Romero

AÑO: 2025

**1.** Introducción y Alcance del Proyecto

El objetivo de esta práctica es desarrollar una aplicación de escritorio en Java que permita gestionar notas académicas de manera local, con interfaz gráfica, persistencia en base de datos y manejo de errores. El sistema permite:

* Registrar notas con título, descripción, calificación, categoría y fecha.
* Visualizar, agregar, editar y eliminar notas.
* Calcular el promedio general de las calificaciones.
* Exportar los datos a un archivo en formato Markdown (.md).
* Persistencia en SQL Server (SQLEXPRESS) con autenticación integrada.

Este sistema cumple con los principios de Programación Orientada a Objetos (POO), modularidad, gestión de excepciones y cohesión, siendo un ejemplo funcional y escalable de una aplicación de dominio académico.

**2.** Modelo de Dominio y Principios de POO

**2.1 Clases del Dominio**

La entidad principal es la clase Nota, que encapsula los datos de una evaluación académica. Las clases del dominio están en el paquete dominio.

Nota.java

* Atributos: id, titulo, descripcion, calificacion, categoria, fecha.
* Métodos: Getters, setters con validación, equals, hashCode, toString.
* Invariantes:
  + calificacion ∈ [0.0, 10.0]
  + titulo no nulo ni vacío
  + categoria ∈ {"Tarea", "Examen", "Proyecto"}
  + fecha no nula

La clase Nota aplica abstracción (oculta detalles de validación) y encapsulamiento (restringe acceso mediante setters validados).

**2.2 Composición vs. Herencia**

* No se utiliza herencia: No existen subtipos naturales de Nota (no hay "NotaFinal", "NotaParcial", etc.).
* Se utiliza composición:
  + GestorNotas tiene un NotaDAO → inyección de dependencia.
  + Esto permite bajo acoplamiento y fácil cambio de implementación (por ejemplo, cambiar de SQL Server a SQLite).

**3.** Arquitectura Modular del Proyecto

El proyecto sigue una arquitectura en capas con paquetes bien definidos:

|  |  |
| --- | --- |
| Paquete | Responsabilidad |
| Servicios | Lógica de aplicación (uso del dominio) |
| Persistencia | Acceso a SQL Server y manejo de JDBC |
| Interfaz | GUI con Swing, eventos y retroalimentación |
| Utilidades | Funcionalidades auxiliares (ej. exportar a |
| Dominio | Modelos y reglas de negocio |

Cohesión alta: Cada paquete tiene una única responsabilidad.  
Bajo acoplamiento: Las capas se comunican mediante interfaces.

**4.** Uso de Estructuras de Datos (Colecciones)

Se utilizan colecciones de java.util para mantener los datos en memoria, sincronizados con la base de datos.

|  |  |  |
| --- | --- | --- |
| Colección | Uso | Justificación |
| ArrayList<Nota> | Almacenar notas en memoria | Acceso secuencial rápido, tamaño dinámico, ideal para listas planas |
| DefaultListModel<String> | Mostrar notas en Jlist | Compatible con Jlist, actualizable en tiempo real |
| HashMap | No usado | No se requiere búsqueda por clave |

Elección óptima: ArrayList es eficiente para iterar y acceder por índice, lo cual es ideal para una lista de notas.

**Conclusión**

Este proyecto cumple con todos los objetivos de aprendizaje:

* POO aplicado con abstracción, encapsulamiento y composición.
* Modularidad claro y bajo acoplamiento.
* GUI funcional con manejo de eventos y validación.
* Estrategia de excepciones bien clasificada.
* Persistencia consistente en SQL Server + archivo opcional.
* Git profesional y documentación clara.