Universidad Autónoma de Tamaulipas

Facultad de Ingeniería Tampico

Ingeniería en Sistemas Computacionales

3o. G

PROGRAMACION AVANZADA

# SISTEMA DE GESTION

Docente: Alvarez Navarro Eduardo

Elaborado por: Brenda Marian Najera Ponce

Resumen del Proyecto: Arquitectura y Patrones de Diseño

Este proyecto implementa una arquitectura genérica de gestión basada en Java Swing, aplicando el patrón Modelo-Vista-Controlador (MVC) y complementándola con tres patrones de diseño adicionales: Singleton, Factory y Strategy. El sistema permite gestionar entidades con atributos definidos dinámicamente por el usuario, y almacenar la información de manera persistente en archivos JSON.

1. Aplicación del Patrón MVC

La arquitectura del proyecto sigue el patrón MVC de forma clara:

- Modelo: Representado por la clase Entidad y sus colecciones, encapsula la lógica de negocio, incluyendo los atributos dinámicos definidos por el usuario y su persistencia en archivos JSON.
- Vista: Se compone de interfaces gráficas construidas con Java Swing, incluyendo ventanas para el menú principal, la creación de entidades, gestión de atributos y edición de registros. Las vistas son genéricas y reutilizables.
- Controlador: Maneja la lógica entre la Vista y el Modelo, respondiendo a eventos del usuario, gestionando la creación de entidades, carga y guardado de archivos, así como la edición de datos.

Esta separación permite una alta mantenibilidad, prueba de componentes por separado y reutilización de código.

2. Patrones de Diseño Implementados

a. Singleton

Se utilizó para la clase GestorEntidades, responsable de administrar todas las entidades del sistema. Esto asegura que exista una única instancia global accesible desde cualquier componente, centralizando la gestión y manteniendo la coherencia de los datos durante la ejecución.

b. Factory

Aplicado en la creación de entidades y atributos. Una Fábrica de Entidades permite generar objetos Entidad configurados con atributos dinámicos en tiempo de ejecución. Este patrón mejora la extensibilidad y facilita la incorporación de nuevos tipos de entidades o validaciones futuras.

c. Strategy

Utilizado para implementar distintas estrategias de validación y visualización de atributos, especialmente útil en el manejo de tipos de datos dinámicos. Por ejemplo,

se pueden aplicar diferentes estrategias para validar campos numéricos, cadenas o fechas, según el tipo definido por el usuario.

3. Gestión de Atributos Dinámicos

Uno de los aspectos más importantes del proyecto es la definición dinámica de atributos:

- El usuario puede crear entidades e ingresar libremente los nombres y tipos de sus atributos.
- Cada Entidad almacena sus atributos en una lista de objetos Atributo (nombre, tipo, valor).
- Los formularios de edición se generan dinámicamente a partir de los atributos definidos, permitiendo CRUD completo sin necesidad de programar cada entidad por separado.
- Esta funcionalidad se integra naturalmente en el MVC: el Modelo almacena y procesa los atributos; la Vista se construye dinámicamente con base en ellos; y el Controlador coordina las operaciones.

4. Persistencia en JSON

Todos los datos (entidades, atributos, registros) se guardan y cargan en archivos JSON. Esto permite:

- Facilidad de lectura y edición externa.
- Portabilidad del sistema.
- Independencia de bases de datos externas.

Este proyecto demuestra cómo construir una aplicación de gestión genérica y extensible utilizando arquitectura MVC en Java Swing, integrando patrones de diseño que aumentan la modularidad, mantenibilidad y escalabilidad. La definición dinámica de atributos otorga gran flexibilidad al sistema, haciéndolo adaptable a distintos dominios sin requerir cambios en la lógica central.