

PROGRAMACION ORIENTADA A OBJETOS

CICLO 2025 ABRIL

INFORME DEL PROYECTO FINAL

DRONEMED – SISTEMA DE GESTION DE DRONES MEDICOS

ALUMNO: Jordi Stefan Cusi Ayala

1. ASPECTOS GENERALES

1.1. Descripción del Proyecto

DroneMed es un sistema de gestión integral para entregas medicas mediante drones autónomos, desarrollado en Java utilizando programación orientada a objetos. El sistema esta diseñado para operar en áreas rurales y de difícil acceso, facilitando la entrega rápida de medicamentos, vacunas y equipos médicos.

1.2. Objetivos del Sistema

- Gestionar una flota de drones con diferentes capacidades de carga
- Administrar pedidos de entregas medicas con sistema de prioridades
- Realizar seguimiento en tiempo real de las entregas
- Gestionar el mantenimiento preventivo y correctivo de los drones
- Proporcionar una interfaz grafica intuitiva para diferentes tipos de usuarios

1.3. Alcance del Proyecto

El sistema abarca:

- Gestión completa de drones (registro, asignación, mantenimiento)
- Sistema de pedidos con priorización (urgente/estándar)
- Visualización de rutas y seguimiento en mapa
- Gestión de clientes (hospitales, centros de salud)
- Reportes de mantenimiento y estadísticas operativas
- Base de datos SQLite para persistencia de información

1.4. Tecnologías Utilizadas

- Lenguaje: JAVA SE

- Interfaz Gráfica: Java Swing
- Base de Datos: SQLite
- Arquitectura Modelo-Vista-Controlador (MVC)
- Paradigma: Programación Orientada a Objetos

2. REQUERIMIENTOS DE LA APLICACIÓN

2.1. Requerimientos Funcionales

2.1.1. Gestión de Drones

- RF01: Registrar drones con especificaciones técnicas (modelo, capacidad, autonomía)
- RF02: Clasificar drones por tipo de carga (ligera, media, pesada)
- RF03: Actualizar estado operativo de drones (disponible, en vuelo, mantenimiento)
- RF04: Asignar drones automáticamente según capacidad y disponibilidad
- RF05: Visualizar ubicación de drones en mapa

2.1.2. Gestión de Pedidos

- RF06: Registrar pedidos de entrega con información del cliente
- RF07: Asignar prioridad a pedidos (urgente, estándar)
- RF08: Calcular tiempo estimado de entrega según tipo de dron
- RF09: Actualizar estado de pedidos (pendiente, asignado, en tránsito, entregado)
- RF10: Generar reportes de entregas

2.1.3. Gestión de Mantenimiento

- RF11: Programar mantenimiento preventivo
- RF12: Registrar mantenimiento correctivo
- RF13: Generar alertas automáticas por mantenimiento requerido
- RF14: Mantener historial de mantenimientos
- RF15: Calcular métricas de disponibilidad de drones

2.1.4. Gestión de Clientes

- RF16: Registrar clientes (hospitales, centros de salud)
- RF17: Calcular distancias automáticamente
- RF18: Mantener historial de pedidos por cliente

2.2. Requerimientos No Funcionales

2.2.1. Usabilidad

- RNF01: Interfaz gráfica intuitiva desarrollada en Java Swing
- RNF02: Navegación clara entre módulos del sistema
- RNF03: Validación de datos en tiempo real

2.2.2. Rendimiento

- RNF04: Tiempo de respuesta menor a 2 segundos para operaciones básicas
- RNF05: Capacidad para gestionar hasta 50 drones simultáneamente
- RNF06: Soporte para múltiples usuarios concurrentes

2.2.3. Confiabilidad

- RNF07: Base de datos SQLite para persistencia confiable
- RNF08: Validación exhaustiva de datos de entrada
- RNF09: Manejo robusto de excepciones

2.2.4. Mantenibilidad

- RNF10: Arquitectura modular basada en patrones de diseño
- RNF11: Código documentado y bien estructurado
- RNF12: Separación clara de responsabilidades (MVC)

3. DISEÑO DE LA APLICACIÓN

3.1. Arquitectura del Sistema

- El sistema DroneMed implementa una arquitectura de tres capas basada en el patrón MVC:
 - 3.1.1. Capa de Presentación (vista)
 - 3.1.2. Capa de Lógica de Negocio (Controlador)
 - 3.1.3. Capa de Datos (Modelo)

3.2. Implementación de Conceptos de POO

- 3.2.1. Herencia
- 3.2.2. Interfaces
- 3.2.3. Polimorfismo
- 3.2.4. Encapsulamiento
- 3.2.5. Abstracción

3.3. Estructura de Paquetes

...

com.dronemed

|—— main/ # Clase principal y configuración

|—— modelo/ # Entidades del dominio
|—— vista/ # Interfaces gráficas
|—— controlador/ # Lógica de negocio
|—— database/ # Acceso a datos
| |—— DAO/ # Data Access Objects
|—— interfaces/ # Interfaces del sistema
|—— utils/ # Utilidades y validaciones
...

3.4. Base de Datos

3.4.1. Esquema de Base de Datos SQLite

Tablas principales:

- drones: Información de drones
- pedidos: Pedidos de entrega
- clientes: Clientes del sistema
- mantenimientos: Registros de mantenimiento
- usuarios: Usuarios del sistema

3.4.2. Patrón DAO

Cada entidad tiene su correspondiente DAO que encapsula las operaciones CRUD:

- DronDAO
- PedidoDAO
- ClienteDAO
- MantenimientoDAO
- UsuarioDAO

3.5. Funcionalidades Implementadas

3.5.1. Gestión de Drones

- Registro de drones con validación de datos
- Clasificación automática por tipo de carga
- Asignación automática basada en capacidad y disponibilidad
- Seguimiento de estado operativo
- Cálculo de métricas de rendimiento

3.5.2. Gestión de Pedidos

- Formulario de registro con validación en tiempo real
- Cálculo automático de distancias
- Estimación de tiempo y costo de entrega
- Sistema de prioridades
- Seguimiento de estado de pedidos

3.5.3. Visualización en Mapa

- Representación gráfica de ubicaciones
- Seguimiento de drones en tiempo real
- Visualización de rutas de entrega
- Indicadores de estado visual

3.5.4. Mantenimiento y Reportes

- Programación de mantenimiento preventivo
- Registro de mantenimiento correctivo
- Generación de reportes estadísticos
- Alertas automáticas

4. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

4.1.1. Cumplimiento de Objetivos

El proyecto DroneMed ha logrado implementar exitosamente un sistema integral de gestión de drones médicos que cumple con todos los requerimientos establecidos.

4.1.2. Implementación de POO

Se han aplicado correctamente todos los conceptos fundamentales de programación orientada a objetos:

- Herencia: Jerarquía de clases Dron con especialización por tipo de carga
- Polimorfismo: Métodos que se comportan diferente según el tipo de objeto
- Encapsulación: Protección de datos y métodos de acceso controlado
- Abstracción: Interfaces y clases abstractas que definen contratos

4.1.3. Arquitectura Robusta

La implementación del patrón MVC proporciona:

- Separación clara de responsabilidades
- Facilidad de mantenimiento y extensión
- Reutilización de componentes
- Testabilidad mejorada

4.1.4. Funcionalidades Completas

El sistema incluye todas las funcionalidades requeridas:

- Gestión completa de drones con diferentes tipos de carga
- Sistema de pedidos con priorización
- Visualización en mapa
- Mantenimiento preventivo y correctivo
- Reportes y estadísticas

4.2. Fortalezas del Sistema

4.2.1. Diseño Orientado a Objetos

- Uso apropiado de herencia para especialización de drones
- Interfaces bien definidas para funcionalidades transversales
- Polimorfismo efectivo en cálculos de tiempo de entrega

4.2.2. Interfaz de Usuario

- Interfaz gráfica intuitiva desarrollada en Java Swing
- Validación de datos en tiempo real
- Navegación clara entre módulos

4.2.3. Persistencia de Datos

- Base de datos SQLite confiable
- Patrón DAO para acceso estructurado a datos
- Inicialización automática de esquema

4.2.4. Manejo de Errores

- Validación exhaustiva de datos de entrada
- Manejo robusto de excepciones
- Mensajes de error informativos

4.3. Recomendaciones para Mejoras Futuras

4.3.1. Funcionalidades Adicionales

- Implementar envío real de emails para notificaciones
- Agregar autenticación y autorización de usuarios

- Desarrollar API REST para integración con sistemas externos
- Implementar seguimiento GPS real de drones

4.3.2. Mejoras Técnicas

- Migrar a base de datos más robusta (PostgreSQL, MySQL)
- Implementar patrón Observer para notificaciones en tiempo real
- Agregar logging más detallado con frameworks
- Implementar pruebas unitarias y de integración

4.3.3. Optimizaciones de Rendimiento

- Implementar cache para consultas frecuentes
- Optimizar consultas de base de datos
- Agregar paginación en listados grandes
- Implementar carga lazy de datos

4.3.4. Mejoras de Usabilidad

- Agregar temas personalizables
- Implementar atajos de teclado
- Mejorar accesibilidad
- Agregar ayuda contextual

4.4. Lecciones Aprendidas

4.4.1. Importancia del Diseño

Un diseño bien planificado facilita significativamente el desarrollo y mantenimiento del sistema.

4.4.2. Valor de los Patrones de Diseño

La implementación de patrones como MVC y DAO proporciona estructura y facilita la comprensión del código.

4.4.3. Beneficios de la POO

La programación orientada a objetos permite crear sistemas más modulares, reutilizables y mantenibles.

4.4.4. Importancia de la Validación

La validación exhaustiva de datos previene errores y mejora la experiencia del usuario.

4.5. Conclusión final

El proyecto DroneMed representa una implementación exitosa de un sistema de gestión de drones médicos que demuestra el dominio de los conceptos de programación orientada a objetos. El sistema es funcional, robusto y extensible, cumpliendo con todos los requerimientos técnicos y funcionales establecidos.

La aplicación de conceptos como herencia, polimorfismo, encapsulación y abstracción ha resultado en un código bien estructurado y mantenible. La arquitectura MVC facilita la separación de responsabilidades y la escalabilidad del sistema.

Este proyecto constituye una base sólida que puede ser extendida con funcionalidades adicionales y optimizaciones según las necesidades futuras del negocio.