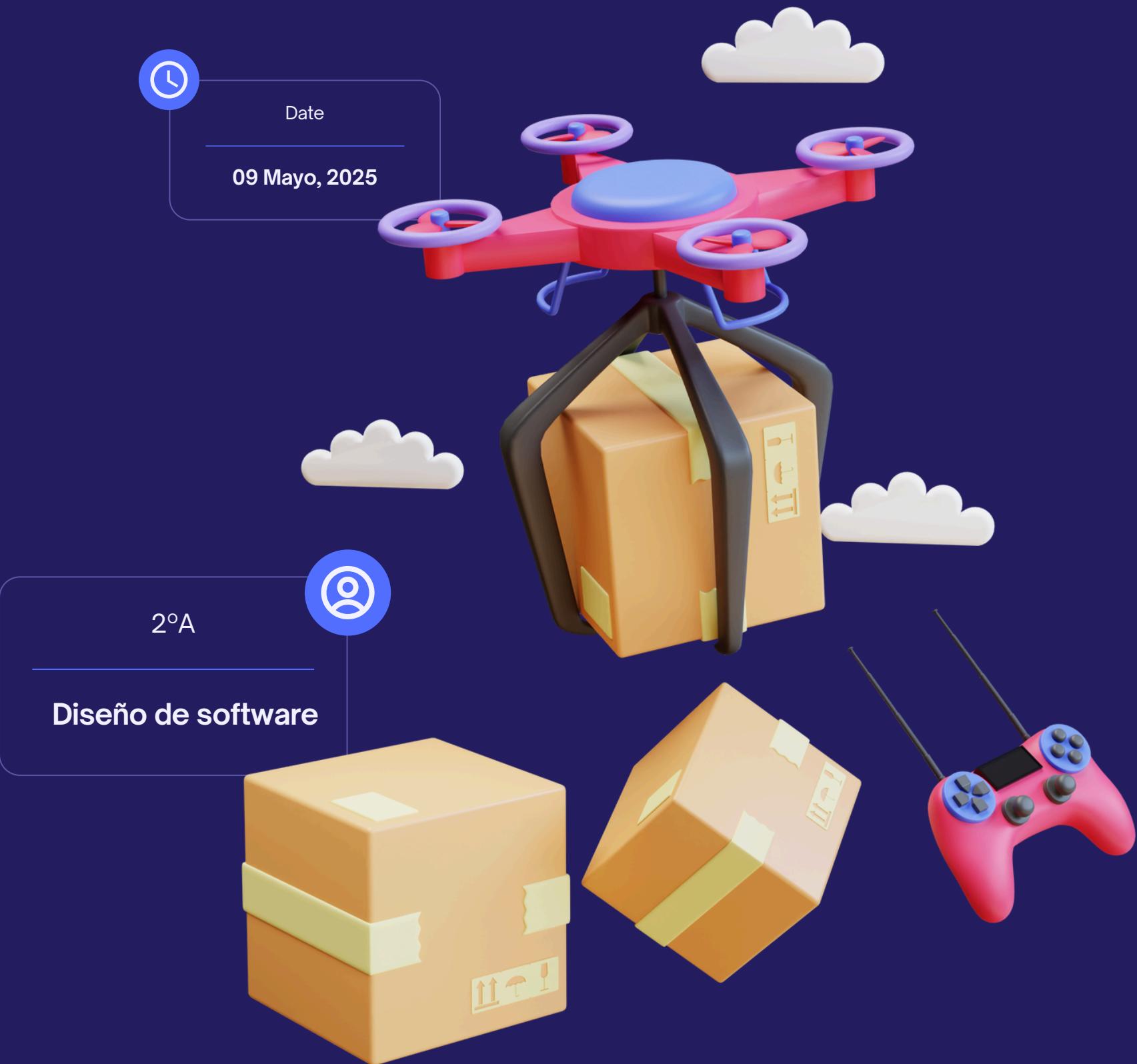


Skyline Logistics

Proyecto final

Luis Marquina, Manuel Martínez, Miguel Torán



1. Introducción al Proyecto

2. Desarrollo y Objetivo

3. Características Principales

4. Patrones de Diseño Utilizados

5. Integración con una API

6. Conclusión

Índice



🧩 Descripción general

Skyline Logistics es un simulador logístico por turnos desarrollado en Java.

El jugador gestiona una empresa de transporte multimodal desde cero, tomando decisiones estratégicas para expandirse y maximizar beneficios.



🕹️ Funcionamiento del programa

Cada jornada se desarrolla por turnos, con aparición de nuevos pedidos, eventos aleatorios (averías, clima, oportunidades) y condiciones de mercado cambiantes.

El sistema apuesta por una arquitectura modular, patrones de diseño robustos y mecánicas que fomentan la toma de decisiones estratégicas, planificación a largo plazo y adaptabilidad.

Introducción al proyecto



Desarrollo del juego

Cada partida es única gracias a la generación aleatoria de:

- Pedidos y clientes
- Vehículos con atributos distintos
- Eventos inesperados
- El jugador puede elegir entre varios modos de juego: libre, campaña o desafío, adaptando la experiencia a distintos estilos.



Objetivo del juego

Diseñar un sistema modular, robusto y mantenible que simule de forma sencilla pero efectiva la gestión logística, aplicando patrones de diseño para resolver problemas reales de arquitectura de software.



Resultados del jugador

- Optimizar beneficios logísticos.
- Tomar decisiones estratégicas en base a recursos limitados.
- Completar pedidos cumpliendo con los requisitos del cliente.
- Adaptarse a condiciones cambiantes y eventos inesperados.

Desarrollo y Objetivo



Características principales



Juego estructurado en turnos diarios, donde se generan nuevos pedidos y oportunidades del mercado.

El jugador gestiona una flota de vehículos (camiones, barcos, aviones), cada uno con características distintas.

```
. Ver flota
. Ver estadísticas
. Pasar al siguiente día
. Finalizar partida

. Seleccione una opción: 2

PEDIOS PENDIENTES:
| | CLIENTE | CARGA | PRIORIDAD | PESO | DESTINO | TIPO | PAGO | ENTREGA |
+---+-----+-----+-----+-----+-----+-----+-----+
|07 | Farmacéutica Pfizer | Vacunas | BAJA | 1580 | Valladolid | REFRIGERADO | $6435 | 13/05/25 |
|85 | Distribuidora de Materiales | Materiales Construcción | NORMAL | 2413 | Valladolid | NORMAL | $5239 | 12/05/25 |
|12 | Distribuidora de Lácteos | Lácteos | BAJA | 1515 | Madrid | PEREcedero | $11147 | 12/05/25 |

Introduzca ID del pedido a gestionar: P707

VEHÍCULOS DISPONIBLES:
|PO | ID | CAPACIDAD | VELOCIDAD | COSTE/KM | SALUD | DESGASTE | CARGAS PERMITIDAS | COSTE TOTAL | FECHA ENTREGA
+---+---+-----+-----+-----+-----+-----+-----+-----+
|01 | C99 | 3513 | 47 | $5 | 100% | 8% | NORMAL, CONGELADO, REFRIGERADO | $2500 | 11/05/25
```



Los pedidos incluyen cargas variadas con necesidades logísticas especiales (perecederos, alto valor, vivos).

Se presentan eventos dinámicos como mantenimiento, averías o condiciones del mercado.

Patrones utilizados



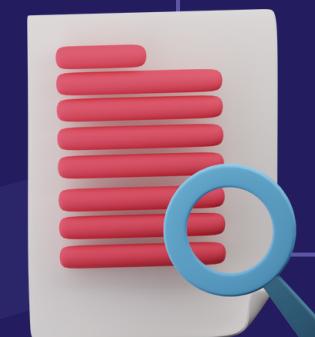
Objetivo de los patrones en Skyline Logistics

En Skyline Logistics se han aplicado múltiples patrones de diseño con el objetivo de estructurar el código de forma modular, extensible y mantenible.

Cada patrón se ha escogido para resolver un problema específico dentro del juego, siguiendo los principios SOLID de la programación orientada a objetos.

Patrones utilizados

Lista de patrones implementados



Patrón Función principal

-  Singleton – Control de estado global (GameManager, recursos).
-  Template Method – Flujos estándar en procesos repetitivos.
-  Strategy – Algoritmos intercambiables según contexto.
-  State – Comportamiento dinámico por estado.
-  Decorator – Añadir mejoras a objetos sin modificar su base.
-  Abstract Factory – Familias de objetos consistentes.
-  Facade – Interfaz simplificada para subsistemas complejos.

Uso en el proyecto

- Se usa en la clase GameManager.
- Garantiza que solo haya una instancia que gestione el estado del juego.

Función

- Coordina flota, pedidos, progreso del jugador, y el modo de juego activo.

Ventajas

- Control centralizado del flujo del juego.
- Evita inconsistencias por múltiples instancias.

Singleton Pattern



Uso en el proyecto

- Implementado en sistemas como misiones, mantenimiento o carga/descarga.
- Define una plantilla de ejecución con pasos que las subclases adaptan.

Función

- Estandariza procesos comunes manteniendo flexibilidad.

Ventajas

- Fomenta la reutilización de lógica base.
- Facilita la incorporación de nuevos tipos de procesos.



Template Method

Uso en el proyecto

- Se aplica para encapsular los diferentes modos de juego.
- Interfaz ModoJuego → implementaciones: ModoLibre, ModoCampaña, ModoDesafío.

Función

- Cada modo define sus propias reglas y condiciones de finalización.

Ventajas

- Permite cambiar el comportamiento del juego en tiempo de ejecución.
- Fomenta el principio abierto/cerrado (Open/Closed).

Strategy Pattern



Uso en el proyecto

- Maneja los estados de vehículos, misiones y empresas (Idle, En curso, Mantenimiento, etc.).
- Cada estado tiene su lógica encapsulada.

Función

- Permite que un objeto cambie su comportamiento según su estado interno.

Ventajas

- Mejora la organización del código.
- Asegura transiciones válidas y controladas.



State Pattern

Uso en el proyecto

- Mejora vehículos con decoradores (motor mejorado, chasis reforzado, etc.).
- Aplica también a misiones o condiciones del entorno.

Función

- Añade funcionalidades de forma dinámica sin modificar la clase base.

Ventajas

- Gran flexibilidad.
- Permite combinar mejoras sin herencia múltiple.

Decorator Pattern



Uso en el proyecto

- Genera familias completas de vehículos o misiones.
- Diferentes fábricas se encargan de producir objetos compatibles.

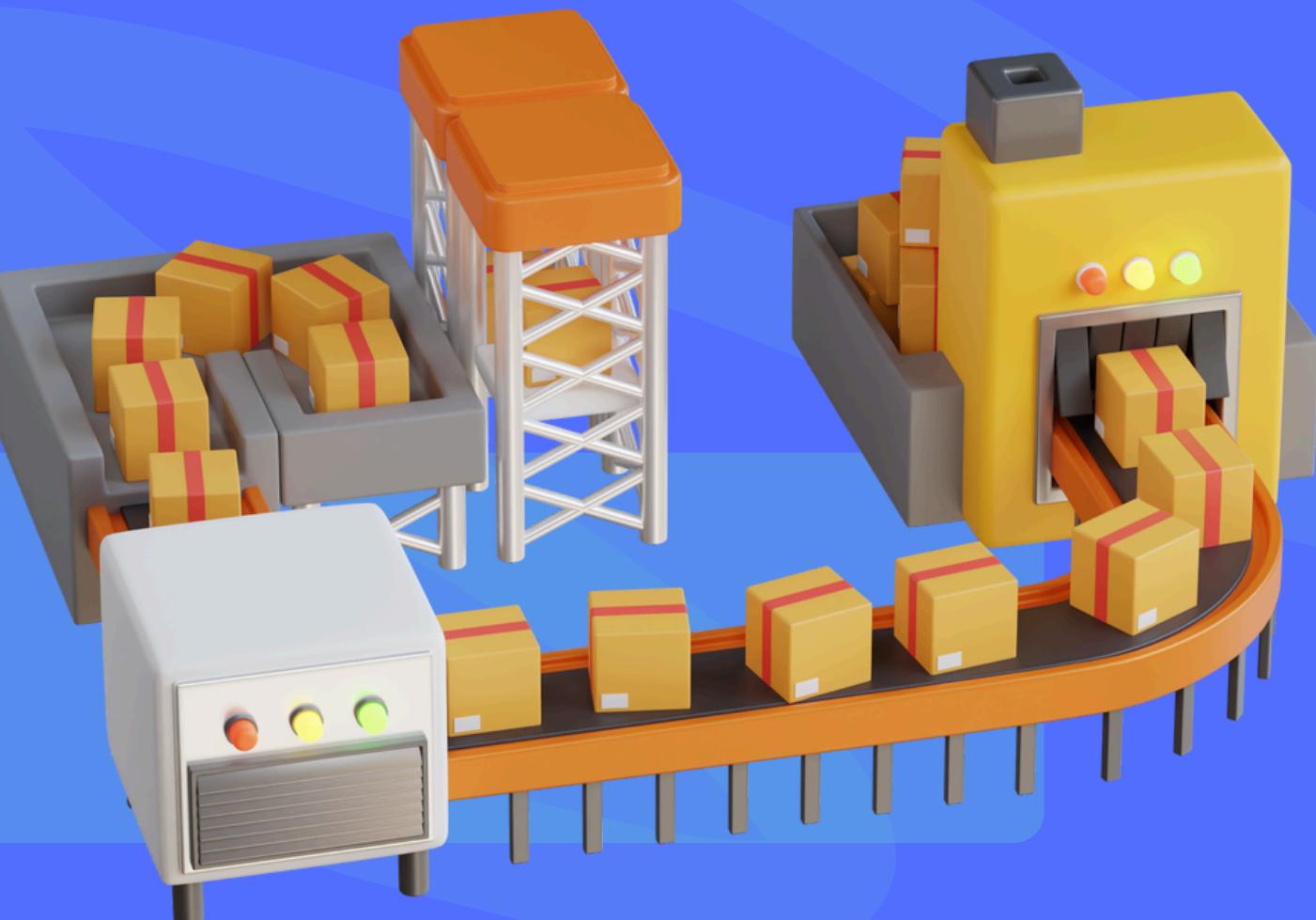
Función

- Ofrece una interfaz para crear objetos sin especificar su clase exacta.

Ventajas

- Promueve la coherencia entre componentes relacionados.
- Permite ampliar tipos sin afectar al código cliente.

Abstract Factory



Uso en el proyecto

- GameFacade y MissionFacade simplifican operaciones como crear misiones o asignar vehículos.
- Ocultan la complejidad de los subsistemas.

Función

- Proporciona una interfaz centralizada y simple para ejecutar procesos complejos.

Ventajas

- Reduce el acoplamiento.
- Mejora la claridad y mantenibilidad del código.

Facade



Strategy + Decorator

Los decoradores pueden modificar las estrategias de movimiento y costo

Permite combinar diferentes comportamientos

State + Template Method

Los estados implementan el template method para procesar pedidos

Cada estado define su propia implementación

Interacciones entre patrones



Abstract Factory + Singleton

La fábrica de vehículos puede ser un singleton

Centraliza la creación de objetos

Facade + Strategy

El facade simplifica el acceso de las siguientes estrategias

Encapsula la complejidad de las operaciones

Integración con una API



Objetivo de la integración

Permitir que Skyline Logistics pueda conectarse o simular conexión con una API externa para enriquecer la experiencia de juego y preparar el sistema para futuras expansiones, como lógica online o sincronización con datos externos.



Ejemplos de integraciones previstas o simuladas

API de mercado de vehículos

Devuelve nuevas ofertas cada día con vehículos únicos.

API de eventos logísticos

Simula condiciones del entorno (clima, tráfico, conflictos).

API climática externa

Afecta tiempos de entrega o disponibilidad de rutas.

API local (mock) para pruebas

Simula peticiones REST para testear lógica de red.

Integración con una API



Tecnologías base

HttpURLConnection o HttpClient de Java para conexión REST.

Serialización/deserialización en JSON (Jackson o Gson).

Clases de utilidad como ApiConnector o RemoteEventFetcher.

Ventajas de la integración

- Añade realismo y profundidad al juego.
- Abre la posibilidad a multijugador o eventos en tiempo real.
- Prepara el sistema para servicios de backend o microservicios.

✓ Valor del proyecto

Skyline Logistics es un sistema funcional, escalable y educativo.

Refleja una sólida comprensión del diseño orientado a objetos. Va más allá del código: simula un dominio real con complejidad progresiva.

📌 Retos superados:

Coordinación entre múltiples clases sin acoplamiento.

Gestión del estado global sin perder consistencia.

Implementación correcta de patrones en contextos reales.

📈 Logros alcanzados:

Se aplicaron 5+ patrones de diseño correctamente.

Arquitectura clara y separación en paquetes.

Rejugabilidad asegurada mediante generación aleatoria.

Modularidad que permite ampliar el sistema fácilmente.

Conclusiones

🔧 Futuras mejoras posibles:

Interfaz gráfica con JavaFX.

Integración completa con APIs públicas.

IA para competencia con empresas rivales.

Visualización de datos con dashboards.

Multiplayer por red local o en la nube.





Tiempo

Tiempo total del proyecto

Hemos tardado entorno a las 80-100H entre todos los integrantes, además de más de 200 commits en Git

Skyline Logistics

Proyecto final

Luis Marquina, Manuel Martínez, Miguel Torán

