TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:

AGOSTO-DICIEMBRE 2025

CARRERA:

INGENIERÍA EN SISTEMAS COMPUTACIONALES / INFORMATICA

MATERIA Y SERIE :

PATRONES DE DISEÑO

TÍTULO:

EXAMEN DE LAS UNIDADES 4 Y 5

UNIDAD A EVALUAR:

4TA y 5TA

NOMBRES Y NÚMEROS DE CONTROL DE LOS ALUMNOS:

ZARZA MORALES JOSE DIEGO 22210366

NOMBRE DEL MAESTRO:

MARIBEL GUERRERO LUIS

FECHA:

09 DE DICIEMBRE DEL 2025

Índice

1.- Código

```
using System;

namespace CapaDominio
{
    public class Entrega
    {
        public string Direccion { get; }
        public string Telefono { get; }
        public string NombreCliente { get; }
        public DateTime FechaSolicitud { get; }
        public EstadoEntrega Estado { get; private set; }

        public Entrega(string direccionDeEntrega, string
telefonoDelCliente, string nombreDelCliente)
        {
            Direccion = ValidarCampoObligatorio(direccionDeEntrega,
"La dirección no puede estar vacía");
            NombreCliente = ValidarCampoObligatorio(nombreDelCliente,
"El nombre del cliente no puede estar vacío");
            Telefono = telefonoDelCliente;
            FechaSolicitud = DateTime.Now;
            Estado = EstadoEntrega.Pendiente;
        }

        private static string ValidarCampoObligatorio(string valor,
string mensajeError)
        {
            if (string.IsNullOrWhiteSpace(valor))
            {
                throw new ArgumentException(mensajeError);
            }
            return valor;
        }

        public void ActualizarEstado(EstadoEntrega nuevoEstado)
        {
            Estado = nuevoEstado;
        }
    }
```

```
    public enum EstadoEntrega
    {
        Pendiente,
        EnProceso,
        Completada,
        Fallida
    }
}
```

```
using System;
using Utilidades;

namespace AlmacenamientoDronesRepartidores.Estados
{
    public interface IEstadoDron
    {
        void Cargar(DronRepartidor dron);
        void Entregar(DronRepartidor dron);
        string ObtenerNombreEstado();
    }

    public class EstadoDescargado : IEstadoDron
    {
        public void Cargar(DronRepartidor dron)
        {
            AyudasVisuales.MostrarTextoConColor($"{dron.IdDelDron}:
Iniciando proceso de carga...", ConsoleColor.Yellow);
            dron.CambiarEstado(new EstadoCargando());
        }

        public void Entregar(DronRepartidor dron)
        {
            Console.WriteLine($"{dron.IdDelDron}: No se puede
entregar. El dron está descargado.");
        }

        public string ObtenerNombreEstado()
        {
            return "Descargado";
        }
```

```
    }

    public class EstadoCargando : IEstadoDron
    {
        public void Cargar(DronRepartidor dron)
        {
            AyudasVisuales.MostrarTextoConColor($"{dron.IdDelDron}:
Cargando bateria...", ConsoleColor.Yellow);
            System.Threading.Thread.Sleep(500);
            AyudasVisuales.MostrarTextoConColor($"{dron.IdDelDron}:
Carga completada al 100%", ConsoleColor.Green);
            dron.CambiarEstado(new EstadoCargado());
        }

        public void Entregar(DronRepartidor dron)
        {
            Console.WriteLine($"{dron.IdDelDron}: No se puede
entregar. El dron está cargando.");
        }

        public string ObtenerNombreEstado()
        {
            return "Cargando";
        }
    }

    public class EstadoCargado : IEstadoDron
    {
        public void Cargar(DronRepartidor dron)
        {
            Console.WriteLine($"{dron.IdDelDron}: El dron ya está
completamente cargado.");
        }

        public void Entregar(DronRepartidor dron)
        {
            AyudasVisuales.MostrarTextoConColor($"{dron.IdDelDron}:
Iniciando entrega...", ConsoleColor.Cyan);
            AyudasVisuales.MostrarTextoConColor($"{dron.IdDelDron}:
Volando hacia el destino...", ConsoleColor.Cyan);
            System.Threading.Thread.Sleep(800);
```

```
            dron.CambiarEstado(new EstadoDescargado());
        }

        public string ObtenerNombreEstado()
        {
            return "Cargado";
        }
    }
}
```

```
using System;
using System.Collections.Concurrent;
using Utilidades;

namespace AlmacenamientoDronesRepartidores
{
    public interface IAlmacenDrones
    {
        DronRepartidor? SacarDeEspera();
        void PonerEnEspera(DronRepartidor dron);
        void MostrarEstadoPool();
    }

    public class DronRepartidor
    {
        public string IdDelDron { get; }
        private Estados.IEstadoDron estadoActual;

        public DronRepartidor(string id)
        {
            IdDelDron = id;
            estadoActual = new Estados.EstadoDescargado();
        }

        public void CambiarEstado(Estados.IEstadoDron nuevoEstado)
        {
            estadoActual = nuevoEstado;
            Console.WriteLine($"{IdDelDron}: Estado cambiado a
{estadoActual.ObtenerNombreEstado()}");
        }
```

```
        private void CambiarEstadoSilencioso(Estados.IEstadoDron
nuevoEstado)
        {
            estadoActual = nuevoEstado;
        }

        public void IniciarCarga() => estadoActual.Cargar(this);
        public void RealizarEntrega() => estadoActual.Entregar(this);
        public string ObtenerEstadoActual() =>
estadoActual.ObtenerNombreEstado();

        public void CargarCompletamente()
        {
            IniciarCarga();
            IniciarCarga();
        }

        public void CargarSilenciosamente()
        {
            CambiarEstadoSilencioso(new Estados.EstadoCargando());
            CambiarEstadoSilencioso(new Estados.EstadoCargado());
        }
    }

    public class AlmacenDrones : IAlmacenDrones
    {
        private const int CapacidadTotalDeDrones = 5;
        private readonly ConcurrentQueue<DronRepartidor>
dronesEnEspera;
        private readonly HashSet<DronRepartidor>
dronesActualmenteEnUso;
        private readonly object bloqueoParaDronesEnUso = new
object();

        public AlmacenDrones(int capacidadInicial =
CapacidadTotalDeDrones)
        {
            dronesEnEspera = new ConcurrentQueue<DronRepartidor>();
            dronesActualmenteEnUso = new HashSet<DronRepartidor>();
            CrearDronesIniciales(capacidadInicial);
```

```
        }

        private void CrearDronesIniciales(int cantidad)
        {
            for (int numeroDron = 1; numeroDron <= cantidad;
numeroDron++)
            {
                var nuevoDron = new
DronRepartidor($"Dron-{numeroDron}");
                nuevoDron.CargarSilenciosamente();
                dronesEnEspera.Enqueue(nuevoDron);
            }
        }

        public DronRepartidor? SacarDeEspera()
        {
            if (dronesEnEspera.TryDequeue(out DronRepartidor?
dronObtenido))
            {
                lock (bloqueoParaDronesEnUso)
                {
                    dronesActualmenteEnUso.Add(dronObtenido);
                }
                return dronObtenido;
            }

            AyudasVisuales.MostrarTextoConColor("No hay drones
disponibles en este momento.", ConsoleColor.Red);
            return null;
        }

        public void PonerEnEspera(DronRepartidor dron)
        {
            if (dron == null) return;

            lock (bloqueoParaDronesEnUso)
            {
                if (!dronesActualmenteEnUso.Remove(dron)) return;
            }

            dron.CargarCompletamente();
```

```
            dronesEnEspera.Enqueue(dron);
            Console.WriteLine($"{dron.IdDelDron} ha sido devuelto a
la estacion.");
        }

        public void MostrarEstadoPool()
        {
            int cantidadDronesEnUso;
            lock (bloqueoParaDronesEnUso)
            {
                cantidadDronesEnUso = dronesActualmenteEnUso.Count;
            }

            AyudasVisuales.MostrarTextoConColor("\nEstado de la
estacion de Drones:", ConsoleColor.Magenta);
            Console.WriteLine($"Drones en espera:
{dronesEnEspera.Count}");
            Console.WriteLine($"Drones en uso:
{cantidadDronesEnUso}");
        }
    }
}
```

```
using System;
using AlmacenamientoDronesRepartidores;
using Utilidades;

namespace CentralDronesRepartidores
{
    public class CentralDronesRepartidores
    {
        private static CentralDronesRepartidores? instanciaUnica;
        private static readonly object bloqueoDeSeguridadParaCreacion
= new object();
        private readonly IAlmacenDrones almacenDeDrones;
        private static readonly Random random = new Random();
        private static readonly string[] mensajesFallo = {
            "[FALLO] El dron no pudo despegar debido a mal clima.",
            "[FALLO] Zona de entrega temporalmente restringida.",
            "[FALLO] El dron detecto obstaculo en la ruta."
```

```
        };

        private CentralDronesRepartidores()
        {
            almacenDeDrones = new AlmacenDrones();
        }

        public static CentralDronesRepartidores ObtenerInstancia()
        {
            if (instanciaUnica == null)
            {
                lock (bloqueoDeSeguridadParaCreacion)
                {
                    if (instanciaUnica == null)
                    {
                        instanciaUnica = new
CentralDronesRepartidores();
                    }
                }
            }
            return instanciaUnica;
        }

        public bool PeticionDeEntrega(string direccion, string
telefono, string cliente, int numeroActual = 0, int totalEntregas =
0)
        {
            if (numeroActual > 0 && totalEntregas > 0)
            {
                AyudasVisuales.MostrarTextoConColor($"\n=== Nueva
Entrega [{numeroActual}/{totalEntregas}] ===", ConsoleColor.Cyan);
            }
            else
            {
                AyudasVisuales.MostrarTextoConColor("\n=== Nueva
Entrega ===", ConsoleColor.Cyan);
            }
            Console.WriteLine($"\nDireccion: {direccion}\nTelefono:
{telefono}\nCliente: {cliente}");

            var dronDisponible = almacenDeDrones.SacarDeEspera();
```

```
            if (dronDisponible == null)
            {
                AyudasVisuales.MostrarTextoConColor("No se pudo
realizar la entrega: no hay drones disponibles.", ConsoleColor.Red);
                return false;
            }

            AyudasVisuales.MostrarTextoConColor($"\nAsignando
{dronDisponible.IdDelDron} para la entrega...", ConsoleColor.Yellow);
            Console.WriteLine($"Estado actual del dron:
{dronDisponible.ObtenerEstadoActual()}");

            dronDisponible.RealizarEntrega();

            bool entregaExitosa =
!direccion.ToLower().Contains("postal");

            if (entregaExitosa)
            {

AyudasVisuales.MostrarTextoConColor($"{dronDisponible.IdDelDron}:
Paquete entregado exitosamente!", ConsoleColor.Green);
                AyudasVisuales.MostrarTextoConColor("\nEntrega
completada!", ConsoleColor.Green);
            }
            else
            {

AyudasVisuales.MostrarTextoConColor($"{dronDisponible.IdDelDron}: No
se pudo completar la entrega.", ConsoleColor.Red);
                AyudasVisuales.MostrarTextoConColor("\nEntrega
fallida!", ConsoleColor.Red);

AyudasVisuales.MostrarTextoConColor(mensajesFallo[random.Next(mensaje
sFallo.Length)], ConsoleColor.Red);
            }

            almacenDeDrones.PonerEnEspera(dronDisponible);
            almacenDeDrones.MostrarEstadoPool();
            return entregaExitosa;
        }
```

```
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using CentralDronesRepartidores;
using CapaDominio;
using Utilidades;

namespace CapaLogicaDeNegocio
{
    public class ServicioEntregas
    {
        private readonly
CentralDronesRepartidores.CentralDronesRepartidores centralDeDrones;

        public ServicioEntregas()
        {
            centralDeDrones =
CentralDronesRepartidores.CentralDronesRepartidores.ObtenerInstancia(
);
        }

        public bool ProcesarSolicitudEntrega(Entrega entrega, int
numeroActual = 0, int totalEntregas = 0)
        {
            try
            {
                entrega.ActualizarEstado(EstadoEntrega.EnProceso);
                bool entregaFueExitosa =
centralDeDrones.PeticionDeEntrega(entrega.Direccion,
entrega.Telefono, entrega.NombreCliente, numeroActual,
totalEntregas);

                FinalizarEntrega(entrega, entregaFueExitosa);
                return entregaFueExitosa;
            }
            catch (Exception error)
            {
```

```
                AyudasVisuales.MostrarTextoConColor($"Error
procesando entrega: {error.Message}", ConsoleColor.Red);
                FinalizarEntrega(entrega, false);
                return false;
            }
        }

        private void FinalizarEntrega(Entrega entrega, bool
fueExitosa)
        {
            if (fueExitosa)
            {
                entrega.ActualizarEstado(EstadoEntrega.Completada);
                SonidosSistema.SonidoExito();
            }
            else
            {
                entrega.ActualizarEstado(EstadoEntrega.Fallida);
                SonidosSistema.SonidoError();
            }
        }
    }

    public class ServicioEstadisticas
    {
        private int contadorDeEntregasTotales;
        private int contadorDeEntregasExitosas;
        private int contadorDeEntregasFallidas;

        public void CargarEstadisticasDesdeHistorial(List<string>
historial)
        {
            foreach (var registro in historial)
            {
                bool esExitosa = registro.Contains("EXITOSA");
                bool esFallida = registro.Contains("FALLIDA");

                if (esExitosa || esFallida)
                {
                    RegistrarEntrega(esExitosa);
                }
```

```
            }
        }

        public void RegistrarEntrega(bool laEntregaFueExitosa)
        {
            contadorDeEntregasTotales++;

            if (laEntregaFueExitosa)
            {
                contadorDeEntregasExitosas++;
            }
            else
            {
                contadorDeEntregasFallidas++;
            }
        }

        public void MostrarEstadisticas()
        {
            AyudasVisuales.MostrarEncabezado("ESTADISTICAS DE
ENTREGAS", ConsoleColor.Green);
            Console.WriteLine($"\n      Total de entregas:
{contadorDeEntregasTotales}");
            AyudasVisuales.MostrarTextoConColor($"      Entregas
exitosas:     {contadorDeEntregasExitosas}", ConsoleColor.Green);
            AyudasVisuales.MostrarTextoConColor($"      Entregas
fallidas:     {contadorDeEntregasFallidas}", ConsoleColor.Red);

            if (contadorDeEntregasTotales > 0)
            {
                double porcentajeDeExito =
(double)contadorDeEntregasExitosas / contadorDeEntregasTotales * 100;
                AyudasVisuales.MostrarTextoConColor($"\n      Tasa de
exito:        {porcentajeDeExito:F2}%", ConsoleColor.Yellow);
            }
            Console.WriteLine();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
```

```
using System.IO;
using Utilidades;

namespace CapaPersistencia
{
    public class RepositorioEntregas
    {
        private readonly List<string> historialDeEntregasEnMemoria =
new List<string>();
        private readonly string rutaDelArchivo =
"historial_entregas.txt";

        public RepositorioEntregas()
        {
            CargarHistorialDesdeDiscoAlIniciar();
        }

        public void GuardarEntrega(string direccion, string
nombreCliente, DateTime fechaDeLaSolicitud, bool fueExitosa)
        {
            string estado;
            if (fueExitosa)
            {
                estado = "EXITOSA";
            }
            else
            {
                estado = "FALLIDA";
            }

            string registroFormateado =
$"{fechaDeLaSolicitud:yyyy-MM-dd HH:mm:ss} | {nombreCliente} |
{direccion} | {estado}";

            historialDeEntregasEnMemoria.Add(registroFormateado);

            try
            {
                File.AppendAllText(rutaDelArchivo, registroFormateado
+ Environment.NewLine);
            }
```

```
            catch (Exception errorAlGuardar)
            {
                AyudasVisuales.MostrarTextoConColor($"Error al
guardar en archivo: {errorAlGuardar.Message}", ConsoleColor.Red);
            }
        }

        private void CargarHistorialDesdeDiscoAlIniciar()
        {
            try
            {
                if (File.Exists(rutaDelArchivo))
                {
                    string[] lineasDelArchivo =
File.ReadAllLines(rutaDelArchivo);

historialDeEntregasEnMemoria.AddRange(lineasDelArchivo);
                }
            }
            catch (Exception errorAlCargar)
            {
                AyudasVisuales.MostrarTextoConColor($"Error al cargar
historial: {errorAlCargar.Message}", ConsoleColor.Red);
            }
        }

        public List<string> ObtenerHistorialCompleto()
        {
            return historialDeEntregasEnMemoria;
        }

        public void MostrarHistorialEnPantalla()
        {
            AyudasVisuales.MostrarEncabezado("HISTORIAL DE ENTREGAS",
ConsoleColor.Yellow);

            bool noHayEntregasRegistradas =
!historialDeEntregasEnMemoria.Any();
            if (noHayEntregasRegistradas)
            {
                AyudasVisuales.MostrarTextoConColor("\n      (No hay
```

```
entregas registradas)\n", ConsoleColor.DarkYellow);
            }
            else
            {
                Console.WriteLine();
                for (int numeroDeEntrega = 0; numeroDeEntrega <
historialDeEntregasEnMemoria.Count; numeroDeEntrega++)
                {
                    int numeroMostrado = numeroDeEntrega + 1;
                    AyudasVisuales.EscribirConColor($"
{numeroMostrado}. ", ConsoleColor.Yellow);

Console.WriteLine($"{historialDeEntregasEnMemoria[numeroDeEntrega]}")
;
                }
                Console.WriteLine();
                AyudasVisuales.MostrarTextoConColor($"      Archivo
guardado en: {rutaDelArchivo}", ConsoleColor.DarkGray);
            }

            AyudasVisuales.MostrarTextoConColor("
═══════════════════════════════════════", ConsoleColor.Yellow);
        }
    }
}
```

```
using System;
using CapaLogicaDeNegocio;
using CapaPersistencia;
using Utilidades;

namespace Fachada
{
    public class SistemaEntregasFachada
    {
        private readonly ServicioEntregas servicioEntregas;
        private readonly RepositorioEntregas repositorio;
        private readonly ServicioEstadisticas estadisticas;

        public SistemaEntregasFachada(ServicioEntregas
servicioEntregas, RepositorioEntregas repositorio,
```

```
ServicioEstadisticas estadisticas)
        {
            this.servicioEntregas = servicioEntregas;
            this.repositorio = repositorio;
            this.estadisticas = estadisticas;
        }

        public void SolicitarEntrega(string direccion, string
telefono, string nombreCliente, int numeroActual = 0, int
totalEntregas = 0)
        {
            var entrega = new CapaDominio.Entrega(direccion,
telefono, nombreCliente);
            bool laEntregaFueExitosa =
servicioEntregas.ProcesarSolicitudEntrega(entrega, numeroActual,
totalEntregas);
            estadisticas.RegistrarEntrega(laEntregaFueExitosa);
            repositorio.GuardarEntrega(direccion, nombreCliente,
entrega.FechaSolicitud, laEntregaFueExitosa);
        }

        public void ProcesarLoteDeEntregas(List<EntregaInfo>
entregas)
        {
            AyudasVisuales.MostrarEncabezado($"Procesando
{entregas.Count} entregas", ConsoleColor.Magenta);

            for (int numeroEntrega = 0; numeroEntrega <
entregas.Count; numeroEntrega++)
            {
                int numeroMostrado = numeroEntrega + 1;
                bool noEsLaUltimaEntrega = numeroEntrega <
entregas.Count - 1;

                var entregaActual = entregas[numeroEntrega];
                SolicitarEntrega(entregaActual.Direccion,
entregaActual.Telefono, entregaActual.NombreCliente, numeroMostrado,
entregas.Count);

                if (noEsLaUltimaEntrega)
                {
```

```
                    System.Threading.Thread.Sleep(300);
                }
            }

            AyudasVisuales.MostrarEncabezado("Lote completado
exitosamente", ConsoleColor.Green);
        }
    }

    public class EntregaInfo
    {
        public string Direccion { get; }
        public string Telefono { get; }
        public string NombreCliente { get; }

        public EntregaInfo(string direccion, string telefono, string
nombreCliente)
        {
            Direccion = direccion;
            Telefono = telefono;
            NombreCliente = nombreCliente;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using Fachada;
using CapaLogicaDeNegocio;
using CapaPersistencia;
using Utilidades;

namespace FlotaDeDronesRepartidores
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.BackgroundColor = ConsoleColor.Black;
            Console.ForegroundColor = ConsoleColor.White;
```

```
            Console.Clear();

            var repositorio = new RepositorioEntregas();
            var servicioEntregas = new ServicioEntregas();
            var estadisticas = new ServicioEstadisticas();

estadisticas.CargarEstadisticasDesdeHistorial(repositorio.ObtenerHist
orialCompleto());
            var sistemaFachada = new
SistemaEntregasFachada(servicioEntregas, repositorio, estadisticas);

            MostrarBienvenida();

            while (true)
            {
                MostrarMenu();
                string opcion = ValidacionesEntrada.LeerEntrada();

                switch (opcion)
                {
                    case "1":
                        Console.Clear();
                        SolicitarEntregaIndividual(sistemaFachada);
                        break;
                    case "2":
                        Console.Clear();
                        SolicitarLoteEntregas(sistemaFachada);
                        break;
                    case "3":
                        Console.Clear();
                        estadisticas.MostrarEstadisticas();
                        break;
                    case "4":
                        Console.Clear();
                        repositorio.MostrarHistorialEnPantalla();
                        break;
                    case "5":
                        Console.Clear();
                        MostrarDespedida();
                        SonidosSistema.SonidoDespedida();
                        return;
```

```
                    default:
                        ValidacionesEntrada.MostrarError("Opcion no
valida. Intente nuevamente.");
                        SonidosSistema.SonidoAdvertencia();
                        break;
                }

                Console.WriteLine("\nPresione cualquier tecla para
continuar...");
                Console.ReadKey();
                Console.Clear();
            }
        }

        static void MostrarBienvenida()
        {
            Console.Clear();
            AyudasVisuales.MostrarTextoConColor("\n
═══════════════════════════════════════════════════════",
ConsoleColor.Cyan);
            AyudasVisuales.MostrarTextoConColor("
", ConsoleColor.Cyan);
            AyudasVisuales.MostrarTextoConColor("
SISTEMA DE ENTREGA POR DRONES                    ",
ConsoleColor.Cyan);
            AyudasVisuales.MostrarTextoConColor("
", ConsoleColor.Cyan);
            AyudasVisuales.MostrarTextoConColor("
═══════════════════════════════════════════════════════",
ConsoleColor.Cyan);
            Console.WriteLine();
        }

        static void MostrarMenu()
        {
            AyudasVisuales.MostrarTextoConColor("\n
═══════════════════════════════════════", ConsoleColor.Gray);
            AyudasVisuales.MostrarTextoConColor("              MENU
PRINCIPAL              ", ConsoleColor.White);
            AyudasVisuales.MostrarTextoConColor("
═══════════════════════════════════════", ConsoleColor.Gray);
```

```
            Console.WriteLine("\n      [1] Solicitar entrega
individual");
            Console.WriteLine("      [2] Procesar lote de entregas");
            Console.WriteLine("      [3] Ver estadisticas");
            Console.WriteLine("      [4] Ver historial de entregas");
            Console.WriteLine("      [5] Salir\n");
            AyudasVisuales.EscribirConColor("    Seleccione una
opcion: ", ConsoleColor.Gray);
        }

        static void MostrarDespedida()
        {
            AyudasVisuales.MostrarTextoConColor("\n
═══════════════════════════════════════════════════════════",
ConsoleColor.Gray);
            AyudasVisuales.MostrarTextoConColor("
Gracias por usar el sistema", ConsoleColor.White);
            AyudasVisuales.MostrarTextoConColor("
Hasta pronto", ConsoleColor.White);
            AyudasVisuales.MostrarTextoConColor("
═══════════════════════════════════════════════════════════",
ConsoleColor.Gray);
            Console.WriteLine();
        }

        static void SolicitarEntregaIndividual(SistemaEntregasFachada
sistemaFachada)
        {
            AyudasVisuales.MostrarEncabezado("NUEVA ENTREGA",
ConsoleColor.Cyan);

            string direccion =
ValidacionesEntrada.LeerTextoObligatorio("Direccion");
            string telefono = ValidacionesEntrada.LeerTelefono();
            string nombre =
ValidacionesEntrada.LeerTextoObligatorio("Nombre del cliente");

            sistemaFachada.SolicitarEntrega(direccion, telefono,
nombre);
        }
```

```
        static void SolicitarLoteEntregas(SistemaEntregasFachada
sistemaFachada)
        {
            AyudasVisuales.MostrarEncabezado("LOTE DE ENTREGAS",
ConsoleColor.Magenta);

            int cantidad =
ValidacionesEntrada.LeerEnteroPositivo("Cuantas entregas desea
procesar?", 1, 100);

            var entregas = new List<EntregaInfo>();

            for (int numero = 1; numero <= cantidad; numero++)
            {
                AyudasVisuales.MostrarTextoConColor($"\n    ---
Entrega {numero} de {cantidad} ---", ConsoleColor.Magenta);

                string direccion =
ValidacionesEntrada.LeerTextoObligatorio("Direccion");
                string telefono = ValidacionesEntrada.LeerTelefono();
                string nombre =
ValidacionesEntrada.LeerTextoObligatorio("Nombre del cliente");

                entregas.Add(new EntregaInfo(direccion, telefono,
nombre));
            }

            Console.WriteLine();
            sistemaFachada.ProcesarLoteDeEntregas(entregas);
        }
    }
}
```

```
using System;
using System.Linq;

namespace Utilidades
{
    public static class AyudasVisuales
    {
```

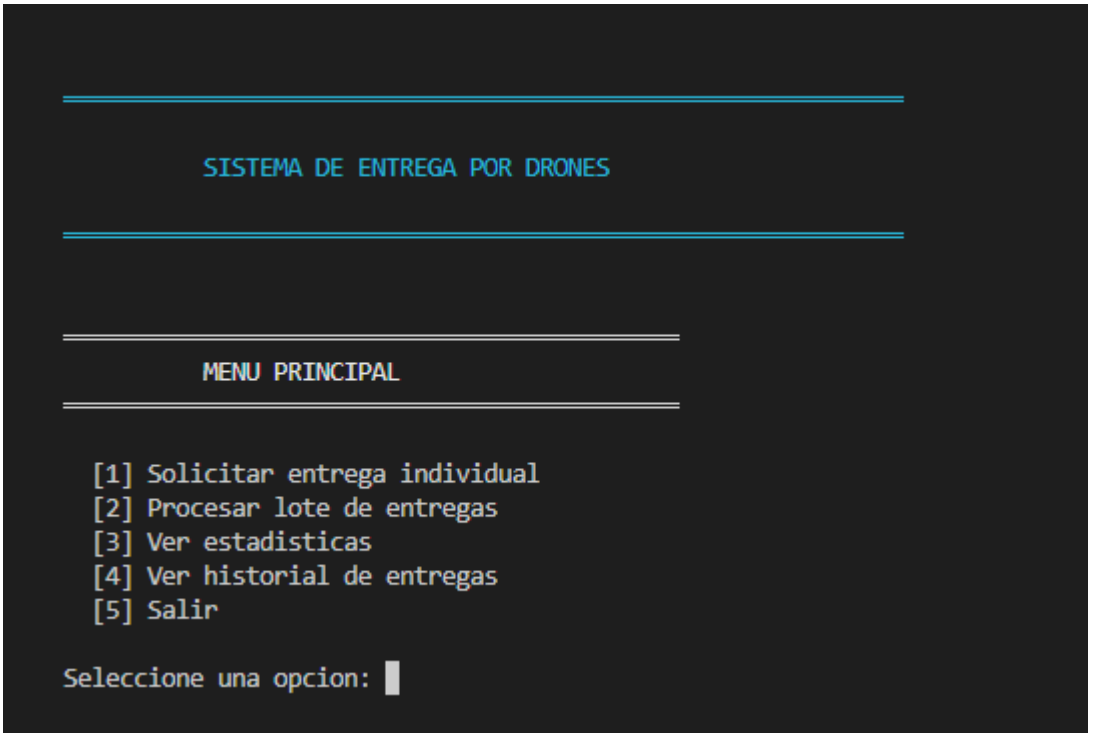
```
        public static void MostrarTextoConColor(string texto,
ConsoleColor color)
        {
            Console.ForegroundColor = color;
            Console.WriteLine(texto);
            Console.ResetColor();
        }

        public static void MostrarLineaDecorativa(string caracter,
int longitud)
        {
            Console.WriteLine(new string(caracter[0], longitud));
        }

        public static void MostrarEncabezado(string titulo,
ConsoleColor color)
        {
            Console.ForegroundColor = color;
            Console.WriteLine($"\n
═══════════════════════════════════════════");
            Console.WriteLine($"                        {titulo}");
            Console.WriteLine("
═══════════════════════════════════════════");
            Console.ResetColor();
        }

        public static void EscribirConColor(string texto,
ConsoleColor color)
        {
            Console.ForegroundColor = color;
            Console.Write(texto);
            Console.ResetColor();
        }
    }

    public static class SonidosSistema
    {
        private static void ReproducirNotas(params (int frecuencia,
int duracion, int pausa)[] notas)
        {
            foreach (var nota in notas)
```

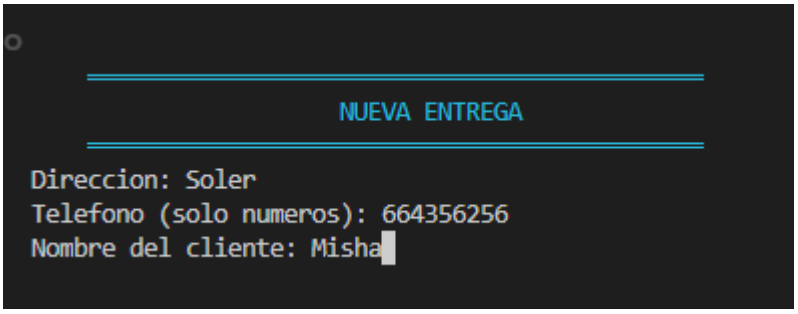
```
            {
                Console.Beep(nota.frecuencia, nota.duracion);
                if (nota.pausa > 0)
                    System.Threading.Thread.Sleep(nota.pausa);
            }
        }

        public static void SonidoExito()
        {
            ReproducirNotas((523, 150, 50), (659, 150, 50), (784,
200, 0));
        }

        public static void SonidoError()
        {
            ReproducirNotas((800, 150, 50), (600, 150, 50), (400,
300, 0));
        }

        public static void SonidoDespedida()
        {
            ReproducirNotas((523, 150, 50), (494, 150, 50), (440,
150, 50), (392, 300, 0));
        }

        public static void SonidoAdvertencia()
        {
            ReproducirNotas((400, 200, 100), (400, 200, 0));
        }
    }

    public static class ValidacionesEntrada
    {
        public static string LeerEntrada()
        {
            string? entrada = Console.ReadLine();
            if (entrada == null)
            {
                return string.Empty;
            }
            else
```

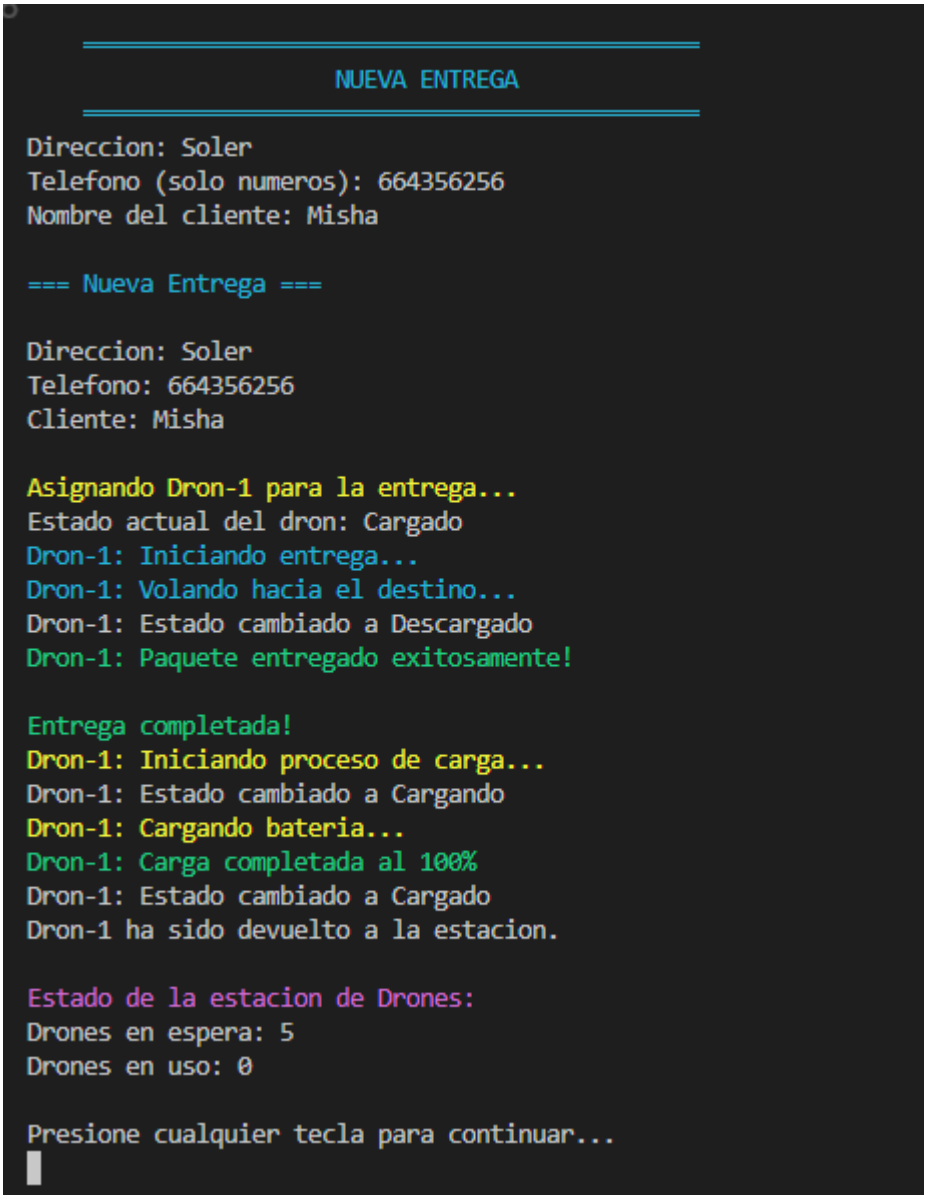
```
            {
                return entrada.Trim();
            }
        }

        public static string LeerTextoObligatorio(string mensaje)
        {
            string valor;
            do
            {
                Console.Write($"{mensaje}: ");
                valor = LeerEntrada();

                if (string.IsNullOrWhiteSpace(valor))
                {
                    MostrarError("Este campo es obligatorio. Intente
nuevamente.");
                }
            } while (string.IsNullOrWhiteSpace(valor));

            return valor;
        }

        public static string LeerTelefono()
        {
            string telefono;
            do
            {
                Console.Write("Telefono (solo numeros): ");
                telefono = LeerEntrada();

                if (string.IsNullOrWhiteSpace(telefono))
                {
                    MostrarError("El telefono es obligatorio.");
                }
                else if (!telefono.All(char.IsDigit))
                {
                    MostrarError("El telefono solo debe contener
numeros.");
                    telefono = string.Empty;
                }
```

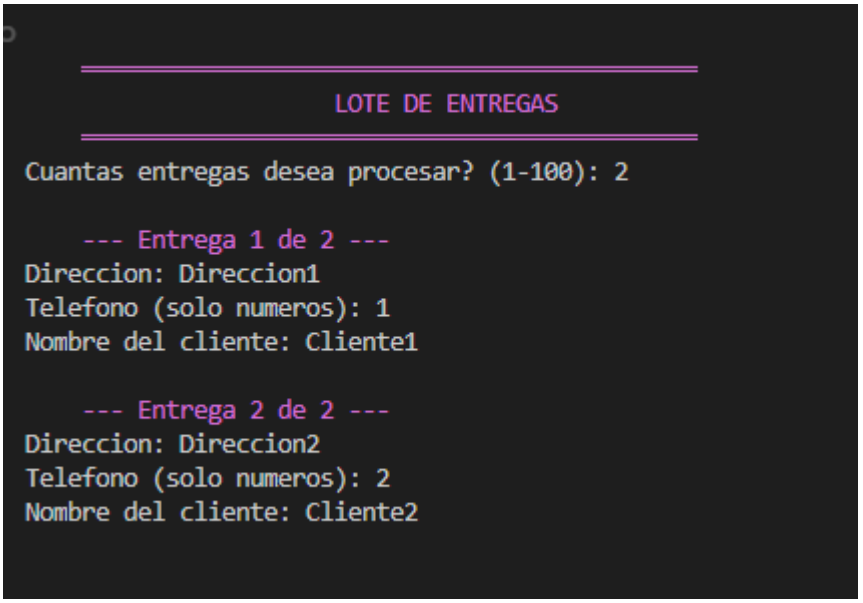
```
            } while (string.IsNullOrWhiteSpace(telefono));

            return telefono;
        }

        public static int LeerEnteroPositivo(string mensaje, int
minimo, int maximo)
        {
            int valor;
            do
            {
                Console.Write($"{mensaje} ({minimo}-{maximo}): ");
                string entrada = LeerEntrada();

                if (!int.TryParse(entrada, out valor))
                {
                    MostrarError("Debe ingresar un numero valido.");
                    continue;
                }

                if (valor < minimo || valor > maximo)
                {
                    MostrarError($"El numero debe estar entre
{minimo} y {maximo}.");
                }
            } while (valor < minimo || valor > maximo || valor == 0);

            return valor;
        }

        public static void MostrarError(string mensaje)
        {
            AyudasVisuales.MostrarTextoConColor($"\n{mensaje}",
ConsoleColor.Red);
        }
    }
}
```

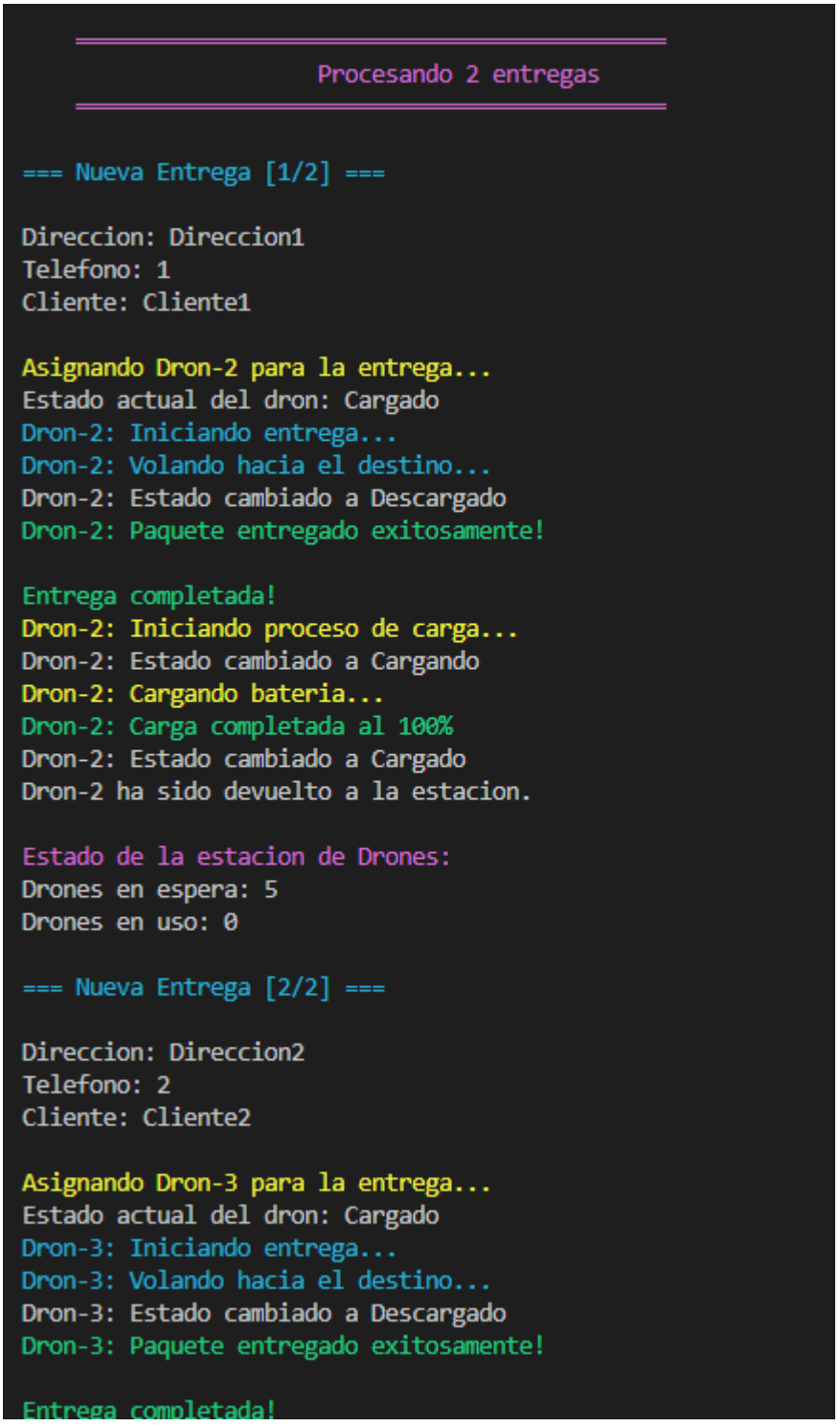
2.-Captura de pantalla de su programa funcionando

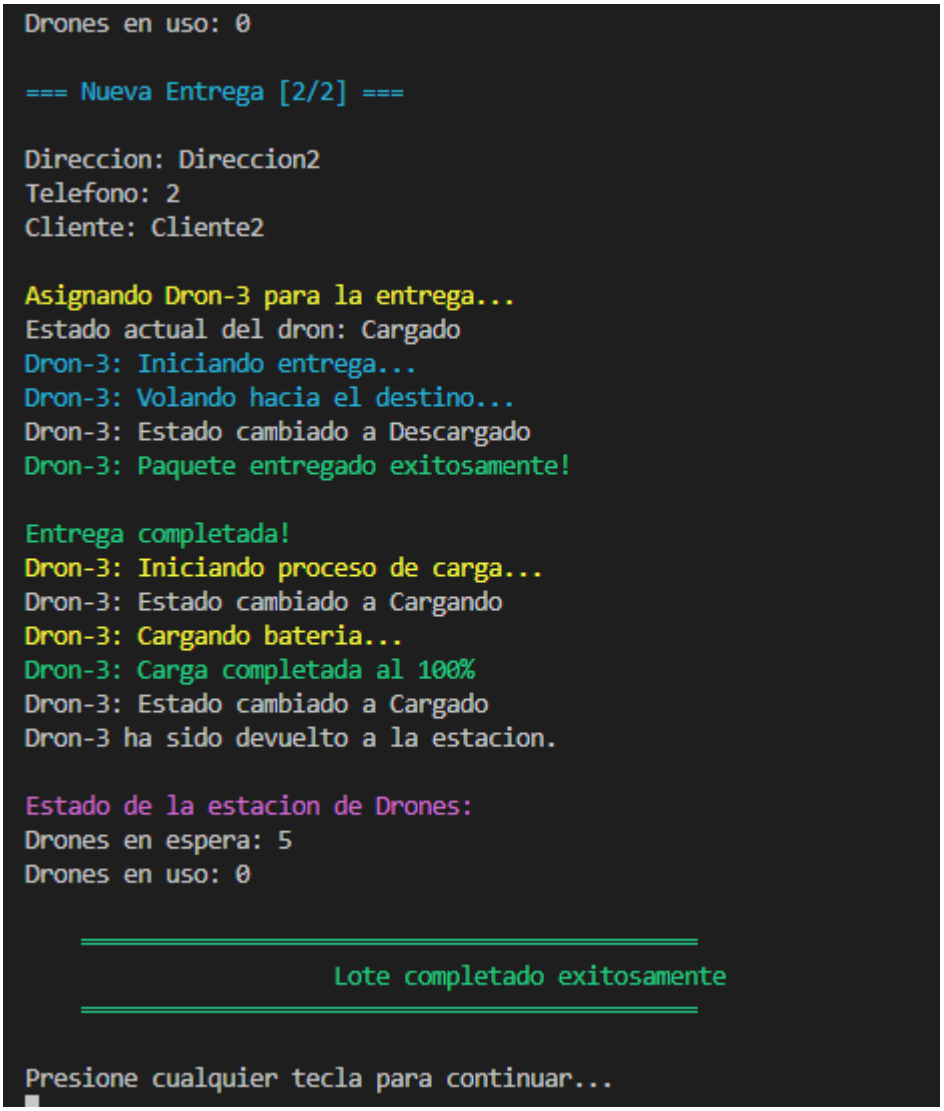


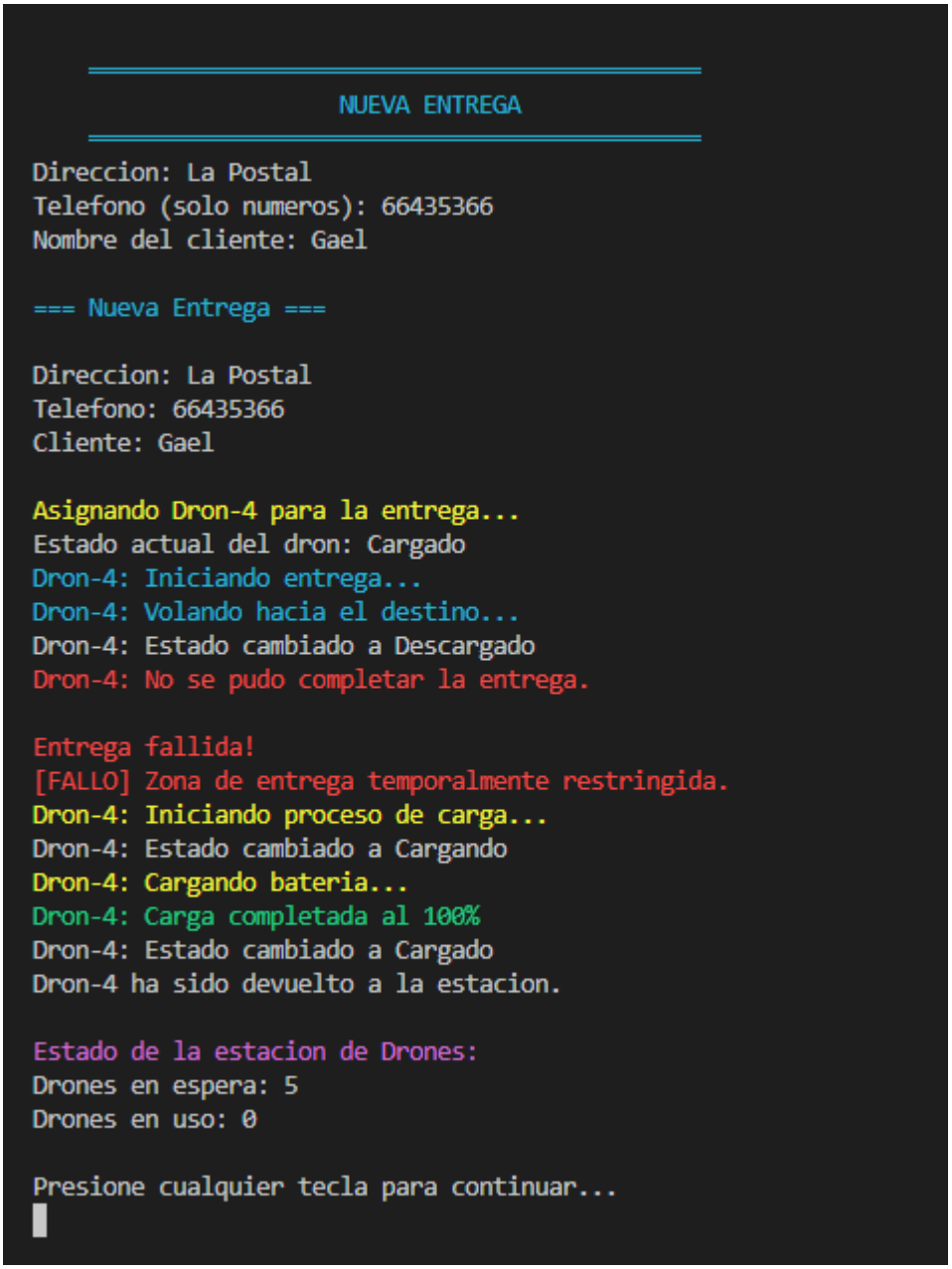


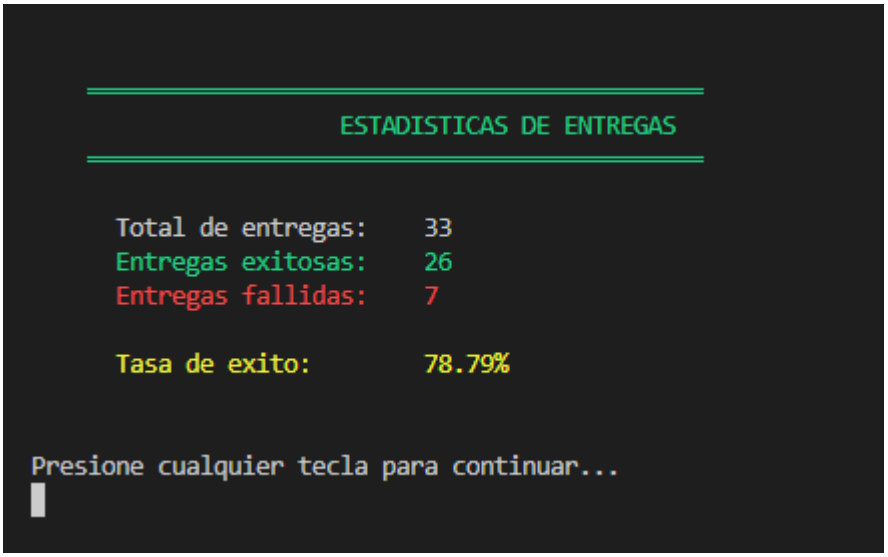

NUEVA ENTREGA
Direccion: Soler
Telefono (solo numeros): 664356256
Nombre del cliente: Misha
=== Nueva Entrega ===
Direccion: Soler
Telefono: 664356256
Cliente: Misha
Asignando Dron-1 para la entrega...
Estado actual del dron: Cargado
Dron-1: Iniciando entrega...
Dron-1: Volando hacia el destino...
Dron-1: Estado cambiado a Descargado
Dron-1: Paquete entregado exitosamente!
Entrega completada!
Dron-1: Iniciando proceso de carga...
Dron-1: Estado cambiado a Cargando
Dron-1: Cargando bateria...
Dron-1: Carga completada al 100%
Dron-1: Estado cambiado a Cargado
Dron-1 ha sido devuelto a la estacion.
Estado de la estacion de Drones:
Drones en espera: 5
Drones en uso: 0
Presione cualquier tecla para continuar...

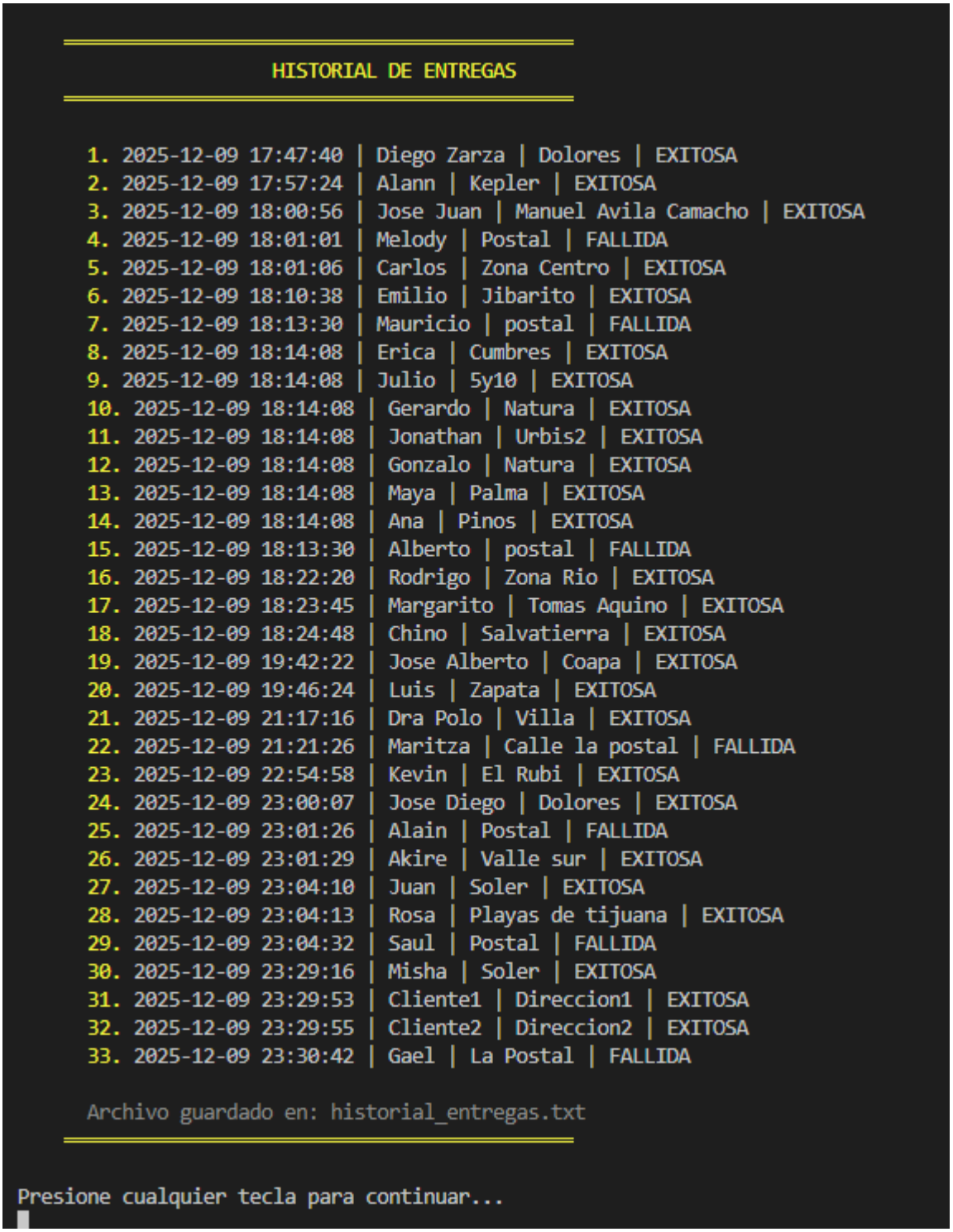

LOTE DE ENTREGAS
Cuantas entregas desea procesar? (1-100): 2
--- Entrega 1 de 2 ---
Direccion: Direccion1
Telefono (solo numeros): 1
Nombre del cliente: Cliente1
--- Entrega 2 de 2 ---
Direccion: Direccion2
Telefono (solo numeros): 2
Nombre del cliente: Cliente2

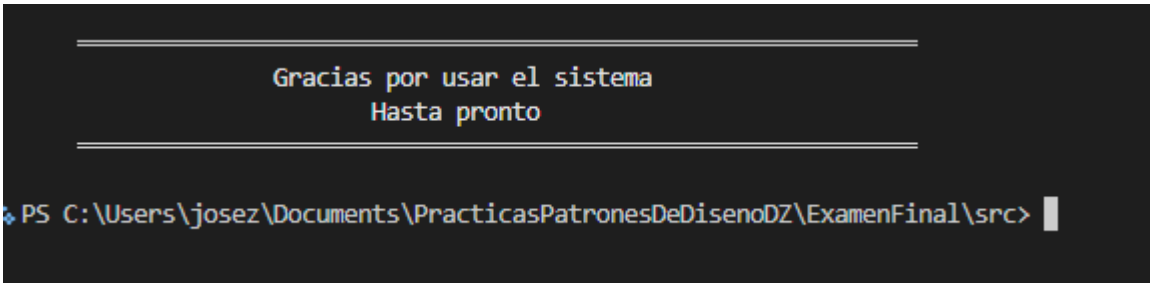

Procesando 2 entregas
=== Nueva Entrega [1/2] ===
Direccion: Direccion1
Telefono: 1
Cliente: Cliente1
Asignando Dron-2 para la entrega...
Estado actual del dron: Cargado
Dron-2: Iniciando entrega...
Dron-2: Volando hacia el destino...
Dron-2: Estado cambiado a Descargado
Dron-2: Paquete entregado exitosamente!
Entrega completada!
Dron-2: Iniciando proceso de carga...
Dron-2: Estado cambiado a Cargando
Dron-2: Cargando bateria...
Dron-2: Carga completada al 100%
Dron-2: Estado cambiado a Cargado
Dron-2 ha sido devuelto a la estacion.
Estado de la estacion de Drones:
Drones en espera: 5
Drones en uso: 0
=== Nueva Entrega [2/2] ===
Direccion: Direccion2
Telefono: 2
Cliente: Cliente2
Asignando Dron-3 para la entrega...
Estado actual del dron: Cargado
Dron-3: Iniciando entrega...
Dron-3: Volando hacia el destino...
Dron-3: Estado cambiado a Descargado
Dron-3: Paquete entregado exitosamente!
Entrega completada!


Drones en uso: 0
=== Nueva Entrega [2/2] ===
Direccion: Direccion2
Telefono: 2
Cliente: Cliente2
Asignando Dron-3 para la entrega...
Estado actual del dron: Cargado
Dron-3: Iniciando entrega...
Dron-3: Volando hacia el destino...
Dron-3: Estado cambiado a Descargado
Dron-3: Paquete entregado exitosamente!
Entrega completada!
Dron-3: Iniciando proceso de carga...
Dron-3: Estado cambiado a Cargando
Dron-3: Cargando bateria...
Dron-3: Carga completada al 100%
Dron-3: Estado cambiado a Cargado
Dron-3 ha sido devuelto a la estacion.
Estado de la estacion de Drones:
Drones en espera: 5
Drones en uso: 0
Lote completado exitosamente
Presione cualquier tecla para continuar...


NUEVA ENTREGA
Direccion: La Postal
Telefono (solo numeros): 66435366
Nombre del cliente: Gael
=== Nueva Entrega ===
Direccion: La Postal
Telefono: 66435366
Cliente: Gael
Asignando Dron-4 para la entrega...
Estado actual del dron: Cargado
Dron-4: Iniciando entrega...
Dron-4: Volando hacia el destino...
Dron-4: Estado cambiado a Descargado
Dron-4: No se pudo completar la entrega.
Entrega fallida!
[FALLO] Zona de entrega temporalmente restringida.
Dron-4: Iniciando proceso de carga...
Dron-4: Estado cambiado a Cargando
Dron-4: Cargando bateria...
Dron-4: Carga completada al 100%
Dron-4: Estado cambiado a Cargado
Dron-4 ha sido devuelto a la estacion.
Estado de la estacion de Drones:
Drones en espera: 5
Drones en uso: 0
Presione cualquier tecla para continuar...


ESTADISTICAS DE ENTREGAS
Total de entregas: 33
Entregas exitosas: 26
Entregas fallidas: 7
Tasa de exito: 78.79%
Presione cualquier tecla para continuar...


HISTORIAL DE ENTREGAS
1. 2025-12-09 17:47:40 | Diego Zarza | Dolores | EXITOSA
2. 2025-12-09 17:57:24 | Alann | Kepler | EXITOSA
3. 2025-12-09 18:00:56 | Jose Juan | Manuel Avila Camacho | EXITOSA
4. 2025-12-09 18:01:01 | Melody | Postal | FALLIDA
5. 2025-12-09 18:01:06 | Carlos | Zona Centro | EXITOSA
6. 2025-12-09 18:10:38 | Emilio | Jibarito | EXITOSA
7. 2025-12-09 18:13:30 | Mauricio | postal | FALLIDA
8. 2025-12-09 18:14:08 | Erica | Cumbres | EXITOSA
9. 2025-12-09 18:14:08 | Julio | 5y10 | EXITOSA
10. 2025-12-09 18:14:08 | Gerardo | Natura | EXITOSA
11. 2025-12-09 18:14:08 | Jonathan | Urbis2 | EXITOSA
12. 2025-12-09 18:14:08 | Gonzalo | Natura | EXITOSA
13. 2025-12-09 18:14:08 | Maya | Palma | EXITOSA
14. 2025-12-09 18:14:08 | Ana | Pinos | EXITOSA
15. 2025-12-09 18:13:30 | Alberto | postal | FALLIDA
16. 2025-12-09 18:22:20 | Rodrigo | Zona Rio | EXITOSA
17. 2025-12-09 18:23:45 | Margarito | Tomas Aquino | EXITOSA
18. 2025-12-09 18:24:48 | Chino | Salvatierra | EXITOSA
19. 2025-12-09 19:42:22 | Jose Alberto | Coapa | EXITOSA
20. 2025-12-09 19:46:24 | Luis | Zapata | EXITOSA
21. 2025-12-09 21:17:16 | Dra Polo | Villa | EXITOSA
22. 2025-12-09 21:21:26 | Maritza | Calle la postal | FALLIDA
23. 2025-12-09 22:54:58 | Kevin | El Rubi | EXITOSA
24. 2025-12-09 23:00:07 | Jose Diego | Dolores | EXITOSA
25. 2025-12-09 23:01:26 | Alain | Postal | FALLIDA
26. 2025-12-09 23:01:29 | Akire | Valle sur | EXITOSA
27. 2025-12-09 23:04:10 | Juan | Soler | EXITOSA
28. 2025-12-09 23:04:13 | Rosa | Playas de tijuana | EXITOSA
29. 2025-12-09 23:04:32 | Saul | Postal | FALLIDA
30. 2025-12-09 23:29:16 | Misha | Soler | EXITOSA
31. 2025-12-09 23:29:53 | Cliente1 | Direccion1 | EXITOSA
32. 2025-12-09 23:29:55 | Cliente2 | Direccion2 | EXITOSA
33. 2025-12-09 23:30:42 | Gael | La Postal | FALLIDA
Archivo guardado en: historial_entregas.txt
Presione cualquier tecla para continuar...

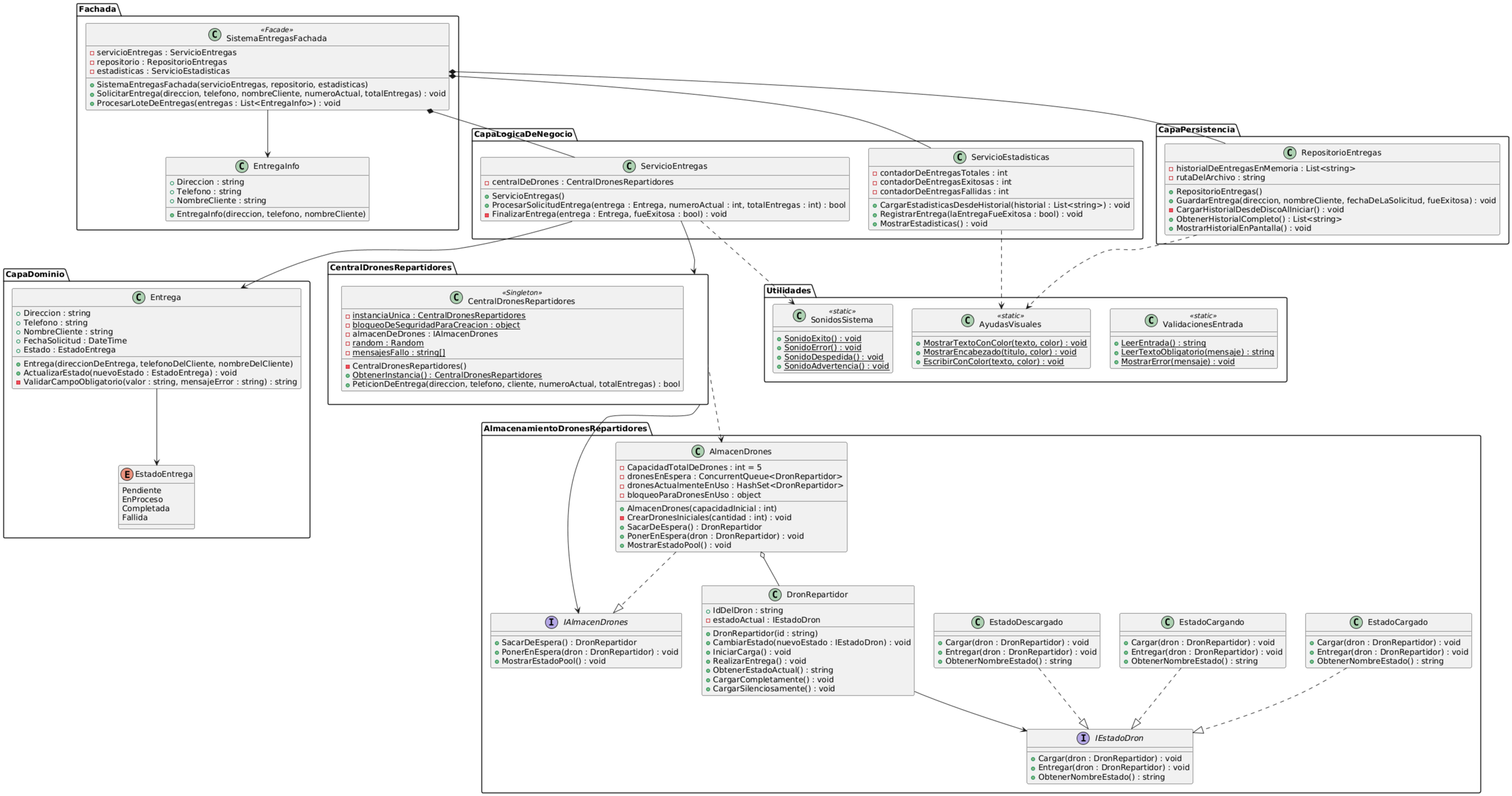

Gracias por usar el sistema
Hasta pronto
PS C:\Users\josez\Documents\PracticasPatronesDeDisenoDZ\ExamenFinal\src>

3.- Conclusión

El uso de estos patrones mejoró bastante la forma en que quedó el sistema. El patrón Singleton evita que se dupliquen recursos importantes, el patrón Piscina de Objetos permite reutilizar objetos que serían costosos de crear cada vez, el patrón Estado elimina condicionales innecesarios y hace el código más claro, el patrón Fachada simplifica todo al dar un punto de acceso único, y la arquitectura por Capas separa bien las responsabilidades y facilita el mantenimiento. Al final, lo más importante que aprendí es que los patrones no funcionan por separado, sino que juntos ayudan a resolver problemas reales y hacen que el código sea más limpio, reutilizable.

4.- Diagrama UML

El Diagrama está abajo jeje


Fachada
«Facade»
SistemaEntregasFachada
- servicioEntregas : ServicioEntregas
- repositorio : RepositorioEntregas
- estadisticas : ServicioEstadisticas
+ SistemaEntregasFachada(servicioEntregas, repositorio, estadisticas)
+ SolicitarEntrega(direccion, telefono, nombreCliente, numeroActual, totalEntregas) : void
+ ProcesarLoteDeEntregas(entregas : List<EntregaInfo>) : void
EntregaInfo
+ Direccion : string
+ Telefono : string
+ NombreCliente : string
+ EntregaInfo(direccion, telefono, nombreCliente)
CapaLogicaDeNegocio
ServicioEntregas
- centralDeDrones : CentralDronesRepartidores
+ ServicioEntregas()
+ ProcesarSolicitudEntrega(entrega : Entrega, numeroActual : int, totalEntregas : int) : bool
- FinalizarEntrega(entrega : Entrega, fueExitosa : bool) : void
ServicioEstadisticas
- contadorDeEntregasTotales : int
- contadorDeEntregasExitosas : int
- contadorDeEntregasFallidas : int
+ CargarEstadisticasDesdeHistorial(historial : List<string>) : void
+ RegistrarEntrega(laEntregaFueExitosa : bool) : void
+ MostrarEstadisticas() : void
CapaPersistencia
RepositorioEntregas
- historialDeEntregasEnMemoria : List<string>
- rutaDelArchivo : string
+ RepositorioEntregas()
+ GuardarEntrega(direccion, nombreCliente, fechaDeLaSolicitud, fueExitosa) : void
- CargarHistorialDesdeDiscoAlIniciar() : void
+ ObtenerHistorialCompleto() : List<string>
+ MostrarHistorialEnPantalla() : void
CapaDominio
Entrega
+ Direccion : string
+ Telefono : string
+ NombreCliente : string
+ FechaSolicitud : DateTime
+ Estado : EstadoEntrega
+ Entrega(direccionDeEntrega, telefonoDelCliente, nombreDelCliente)
+ ActualizarEstado(nuevoEstado : EstadoEntrega) : void
- ValidarCampoObligatorio(valor : string, mensajeError : string) : string
EstadoEntrega
Pendiente
EnProceso
Completada
Fallida
CentralDronesRepartidores
«Singleton»
CentralDronesRepartidores
- instanciaUnica : CentralDronesRepartidores
- bloqueoDeSeguridadParaCreacion : object
- almacenDeDrones : IAlmacenDrones
- random : Random
- mensajesFallo : string[]
- CentralDronesRepartidores()
+ ObtenerInstancia() : CentralDronesRepartidores
+ PeticionDeEntrega(direccion, telefono, cliente, numeroActual, totalEntregas) : bool
Utilidades
«static»
SonidosSistema
+ SonidoExito() : void
+ SonidoError() : void
+ SonidoDespedida() : void
+ SonidoAdvertencia() : void
«static»
AyudasVisuales
+ MostrarTextoConColor(texto, color) : void
+ MostrarEncabezado(titulo, color) : void
+ EscribirConColor(texto, color) : void
«static»
ValidacionesEntrada
+ LeerEntrada() : string
+ LeerTextoObligatorio(mensaje) : string
+ MostrarError(mensaje) : void
AlmacenamientoDronesRepartidores
AlmacenDrones
- CapacidadTotalDeDrones : int = 5
- dronesEnEspera : ConcurrentQueue<DronRepartidor>
- dronesActualmenteEnUso : HashSet<DronRepartidor>
- bloqueoParaDronesEnUso : object
+ AlmacenDrones(capacidadInicial : int)
- CrearDronesIniciales(cantidad : int) : void
+ SacarDeEspera() : DronRepartidor
+ PonerEnEspera(dron : DronRepartidor) : void
+ MostrarEstadoPool() : void
IAlmacenDrones
+ SacarDeEspera() : DronRepartidor
+ PonerEnEspera(dron : DronRepartidor) : void
+ MostrarEstadoPool() : void
DronRepartidor
+ IdDelDron : string
- estadoActual : IEstadoDron
+ DronRepartidor(id : string)
+ CambiarEstado(nuevoEstado : IEstadoDron) : void
+ IniciarCarga() : void
+ RealizarEntrega() : void
+ ObtenerEstadoActual() : string
+ CargarCompletamente() : void
+ CargarSilenciosamente() : void
EstadoDescargado
+ Cargar(dron : DronRepartidor) : void
+ Entregar(dron : DronRepartidor) : void
+ ObtenerNombreEstado() : string
EstadoCargando
+ Cargar(dron : DronRepartidor) : void
+ Entregar(dron : DronRepartidor) : void
+ ObtenerNombreEstado() : string
EstadoCargado
+ Cargar(dron : DronRepartidor) : void
+ Entregar(dron : DronRepartidor) : void
+ ObtenerNombreEstado() : string
IEstadoDron
+ Cargar(dron : DronRepartidor) : void
+ Entregar(dron : DronRepartidor) : void
+ ObtenerNombreEstado() : string