

Universidad ORT Uruguay
Facultad de Ingeniería
Escuela de Tecnología

OBLIGATORIO PROGRAMACION 2

Grupo M2A



Agustín Butrico – 339579



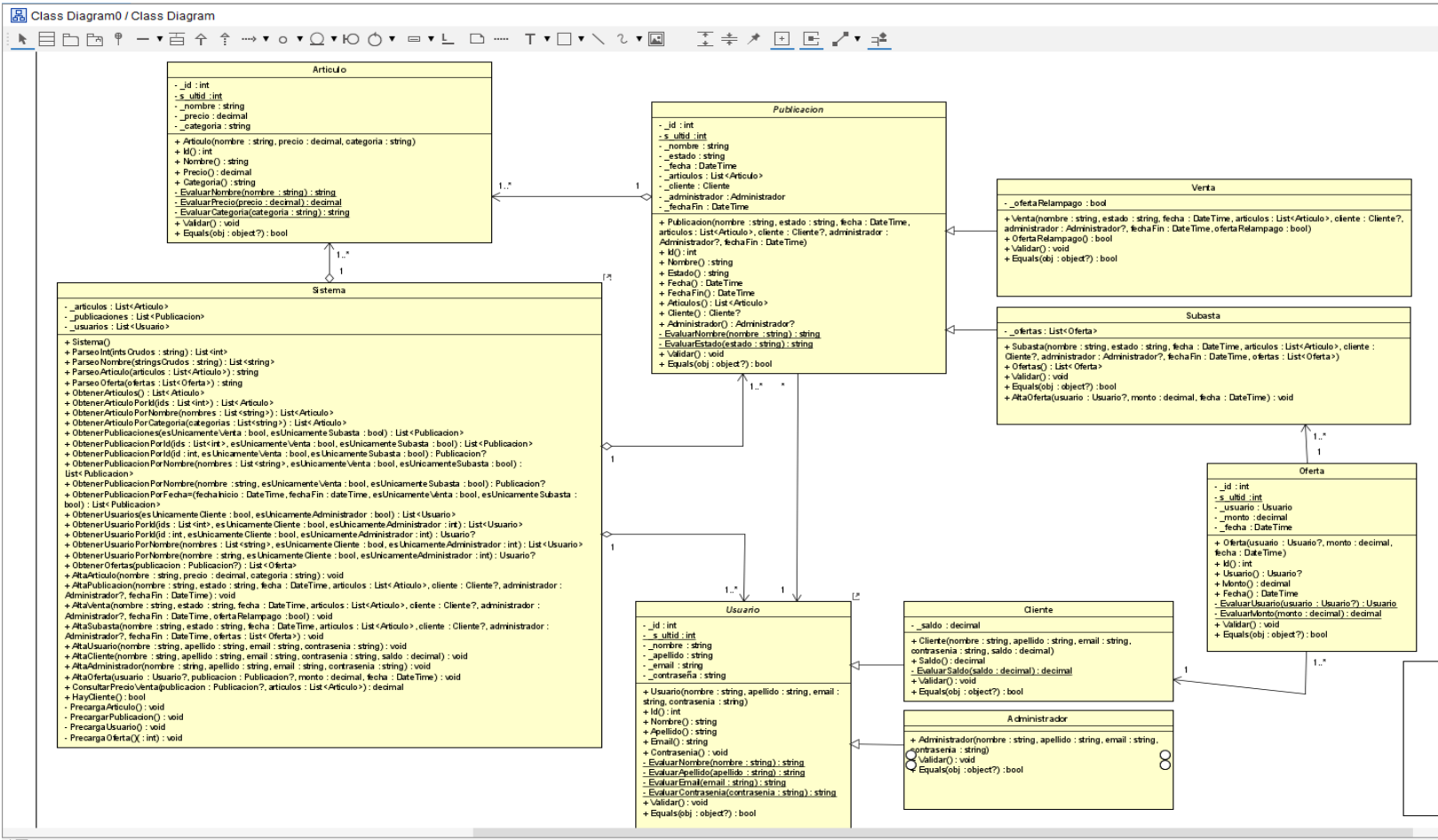
Valentín Latorre – 339103

Profesora de la materia: Liliana Pino

Indice

Diagrama de Clases completo del Dominio del problema	3
Tablas de datos Precargados.....	4
Consultas realizadas a ChatGPT	9
PROGRAM.CS	11
ADMINISTRADOR.CS	66
ARTICULO.CS.....	68
CLIENTE.CS	72
OFERTA.CS	75
PUBLICACION.CS	78
SISTEMA.CS.....	83
SUBASTA.CS	118
USUARIO.CS	121
VENTA.CS	125

Diagrama de Clases completo del Dominio del problema



Tablas de datos Precargados

PrecargarPublicacion()
AltaVenta("Verano en la playa", "ABIERTA", DateTime.ParseExact("05/01/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 11, 24, 35, 54 }}, null, null, DateTime.MinValue, false);
AltaSubasta("Vuelta ciclista", "ABIERTA", DateTime.ParseExact("06/01/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 27, 33, 39 }}, null, null, DateTime.MinValue, new List<Oferta>());
AltaSubasta("Set camping", "ABIERTA", DateTime.ParseExact("21/07/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 7, 34, 36 }}, null, null, DateTime.MinValue, new List<Oferta>());
AltaVenta("Juego gimnasio", "ABIERTA", DateTime.ParseExact("13/12/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 14, 15, 25, 26, 28, 38 }}, null, null, DateTime.MinValue, false);
AltaVenta("Caminata en el bosque", "ABIERTA", DateTime.ParseExact("12/02/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 1, 3, 4, 5 }}, null, null, DateTime.MinValue, false);
AltaVenta("Paseo en bicicleta", "ABIERTA", DateTime.ParseExact("15/03/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 6, 8, 9, 10 }}, null, null, DateTime.MinValue, false);
AltaVenta("Clase de yoga", "ABIERTA", DateTime.ParseExact("22/04/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 12, 13, 16, 18, 20 }}, null, null, DateTime.MinValue, false);
AltaVenta("Día de spa", "ABIERTA", DateTime.ParseExact("30/05/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 21, 22, 23, 29 }}, null, null, DateTime.MinValue, false);
AltaVenta("Concierto al aire libre", "ABIERTA", DateTime.ParseExact("01/08/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 30, 31, 32, 34, 37 }}, null, null, DateTime.MinValue, false);
AltaVenta("Cata de vinos", "ABIERTA", DateTime.ParseExact("10/09/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 40, 41, 42 }}, null, null, DateTime.MinValue, false);
AltaVenta("Taller de pintura", "ABIERTA", DateTime.ParseExact("15/10/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 43, 44, 45, 46 }}, null, null, DateTime.MinValue, false);
AltaVenta("Excursión a la montaña", "ABIERTA", DateTime.ParseExact("25/11/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 47, 48, 49 }}, null, null, DateTime.MinValue, false);
AltaSubasta("Torneo de ajedrez", "ABIERTA", DateTime.ParseExact("12/03/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 50, 51, 52 }}, null, null, DateTime.MinValue, new List<Oferta>());
AltaSubasta("Subasta de arte", "ABIERTA", DateTime.ParseExact("20/04/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 51, 53, 54 }}, null, null, DateTime.MinValue, new List<Oferta>());
AltaSubasta("Rally de coches", "ABIERTA", DateTime.ParseExact("01/06/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 36, 37, 38 }}, null, null, DateTime.MinValue, new List<Oferta>());

AltaSubasta("Subasta de antigüedades", "ABIERTA", DateTime.ParseExact("15/07/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 29, 20, 21 }), null, null, DateTime.MinValue, new List<Oferta>());
AltaSubasta("Concurso de cocina", "ABIERTA", DateTime.ParseExact("05/08/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 42, 43, 44 }), null, null, DateTime.MinValue, new List<Oferta>());
AltaSubasta("Maratón de lectura", "ABIERTA", DateTime.ParseExact("12/09/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 45, 46, 47 }), null, null, DateTime.MinValue, new List<Oferta>());
AltaSubasta("Competencia de fotografía", "ABIERTA", DateTime.ParseExact("30/10/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 18, 19, 20 }), null, null, DateTime.MinValue, new List<Oferta>());
AltaSubasta("Fiesta de disfraces", "ABIERTA", DateTime.ParseExact("15/11/2024", "dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 21, 22, 23 }), null, null, DateTime.MinValue, new List<Oferta>());

PrecargaUsuario()
AltaAdministrador("Valentin", "Latorre", "ValentinLatorre@Gmail.com", "Valentin1234");
AltaAdministrador("Agustin", "Butrico", "AgustinButrico@gmail.com", "Agustin1234");
AltaCliente("Juan", "Peres", "Juanperes@hmail.com", "Juan1234", 2000);
AltaCliente("Esteban", "Lopez", "EstebanLopez@hmail.com", "556643", 2000);
AltaCliente("Carlos", "Medina", "CarlosMedina@hmail.com", "Medina1234", 4500);
AltaCliente("Mariano", "Morales", "MarianoMorales@hmail.com", "Mariano2", 5000);
AltaCliente("Estela", "Rosales", "EstelaRosales@hmail.com", "Rosalia46", 300);
AltaCliente("Marcos", "Sauce", "MarcosSauce@hmail.com", "Sauce31", 30000);
AltaCliente("Lucia", "Gomez", "LuciaGomezs@hmail.com", "Lucia1990", 7200);
AltaCliente("Rodrigo", "Barrios", "RodrigoBarrios@hmail.com", "RodrigoBarrios12", 900);

PrecargaOferta()

AltaOferta(ObtenerUsuarioPorId(3, true, false), ObtenerPublicacionPorId(2, false, true), 100, DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

AltaOferta(ObtenerUsuarioPorId(6, true, false), ObtenerPublicacionPorId(1, false, true), 1500, DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

AltaOferta(ObtenerUsuarioPorId(4, true, false), ObtenerPublicacionPorId(1, false, true), 3400, DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

AltaOferta(ObtenerUsuarioPorId(8, true, false), ObtenerPublicacionPorId(1, false, true), 3500, DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

AltaOferta(ObtenerUsuarioPorId(3, true, false), ObtenerPublicacionPorId(1, false, true), 20000, DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

PrecargaArticulo()

AltaArticulo("Pelota de football", 450, "Football");
AltaArticulo("Camiseta deportiva", 1200, "Deporte");
AltaArticulo("Zapatillas trekking", 3500, "Treking");
AltaArticulo("Raqueta de tenis", 4200, "Tenis");
AltaArticulo("Balón de basquetball", 800, "Basquetball");
AltaArticulo("Guantes de boxeo", 2200, "Boxeo");
AltaArticulo("Casco de ciclismo", 1800, "Ciclismo");
AltaArticulo("Saco de dormir", 2300, "Camping");
AltaArticulo("Bolsa de gimnasio", 950, "Boxeo");
AltaArticulo("Bicicleta de montaña", 15000, "Ciclismo");
AltaArticulo("Mochila de trekking", 2100, "Treking");
AltaArticulo("Protector solar", 320, "Playa");
AltaArticulo("Botella térmica", 750, "Camping");
AltaArticulo("Palo de hockey", 1700, "Hokey");
AltaArticulo("Pesas ajustables", 3000, "Gimnasio");
AltaArticulo("Cinta para correr", 25000, "Gimnasio");
AltaArticulo("Guantes de arquero", 900, "Arquería");
AltaArticulo("Tabla de surf", 12000, "Surf");
AltaArticulo("Canilleras", 600, "Football");
AltaArticulo("Traje de neopreno", 5400, "Surf");
AltaArticulo("Gafas de natación", 650, "Natación");
AltaArticulo("Bola de bowling", 3500, "Bowling");
AltaArticulo("Skateboard", 2400, "Skating");
AltaArticulo("Patines en línea", 2900, "Patinaaje");
AltaArticulo("Salvavidas", 1200, "Playa");
AltaArticulo("Set de pesas", 4200, "Gimnasio");
AltaArticulo("Cuerda para saltar", 300, "Gimnasio");
AltaArticulo("Bicicleta de carrera", 18500, "Ciclismo");
AltaArticulo("Tobilleras con peso", 850, "Gimnasio");
AltaArticulo("Set de dardos", 400, "Juegos");
AltaArticulo("Bate de baseball", 1900, "Baseball");
AltaArticulo("Bola de volleyball", 850, "Volleyball");
AltaArticulo("Aro de basquetball", 2700, "Basquetball");
AltaArticulo("Zapatilla de ciclismo", 1900, "Ciclismo");
AltaArticulo("Silla de camping", 1100, "Camping");
AltaArticulo("Sombrilla", 1600, "Playa");
AltaArticulo("Tienda de campaña", 8700, "Camping");
AltaArticulo("Colchoneta de yoga", 1200, "Deporte");
AltaArticulo("Barra de dominadas", 1900, "Gimnasio");
AltaArticulo("Malla", 600, "Ciclismo");
AltaArticulo("Reloj deportivo", 6500, "Deporte");
AltaArticulo("Monopatín eléctrico", 18000, "Ciclismo");
AltaArticulo("Kit de pesca", 3200, "Pesca");
AltaArticulo("Bolsa de golf", 7600, "Golf");
AltaArticulo("Raqueta de badminton", 1600, "Badminton");
AltaArticulo("Patineta longboard", 3300, "Skating");
AltaArticulo("Bola de rugby", 1050, "Rugby");
AltaArticulo("Kit de snorkel", 1800, "Natacion");
AltaArticulo("Camiseta de compresión", 1300, "Deporte");
AltaArticulo("Gorra deportiva", 400, "Deporte");
AltaArticulo("Balón medicinal", 2000, "Salud");

AltaArticulo("Kit de arquería", 9800, "Arquería");
AltaArticulo("Soga de escalada", 5600, "Escalada");
AltaArticulo("Casco de ski", 3700, "Ski");
AltaArticulo("Balde", 1050, "Playa");
AltaArticulo("Gafas de ciclismo", 900, "Ciclismo");

Consultas realizadas a ChatGPT

Para la generación de los 50 artículos

Me podrías generar una lista de 50 artículos que tenga Nombre y Precio, en este formato :
AltaArticulo("Pelota", 450);
Con valores "realistas"



Aquí tienes una lista de 50 artículos con nombres y precios realistas, en el formato que solicitaste:

csharp

Copiar código

```
AltaArticulo("Pelota de fútbol", 450);  
AltaArticulo("Camiseta deportiva", 1200);  
AltaArticulo("Zapatillas running", 3500);  
AltaArticulo("Raqueta de tenis", 4200);  
AltaArticulo("Balón de baloncesto", 800);  
AltaArticulo("Guantes de boxeo", 2200);  
AltaArticulo("Casco de ciclismo", 1800);  
AltaArticulo("Saco de dormir", 2300);  
AltaArticulo("Bolsa de gimnasio", 950);  
AltaArticulo("Bicicleta de montaña", 15000);  
AltaArticulo("Mochila de trekking", 2100);  
AltaArticulo("Botella térmica", 750);  
AltaArticulo("Palo de hockey", 1700);  
AltaArticulo("Pesas ajustables", 3000);  
AltaArticulo("Cinta para correr", 25000);  
AltaArticulo("Guantes de arquero", 900);  
AltaArticulo("Tabla de surf", 12000);  
AltaArticulo("Canilleras", 600);  
AltaArticulo("Traje de neopreno", 5400);  
AltaArticulo("Gafas de natación", 650);  
AltaArticulo("Bola de bowling", 3500);  
AltaArticulo("Skateboard", 2400);  
AltaArticulo("Patines en línea", 2900);  
AltaArticulo("Set de pesas", 4200);  
AltaArticulo("Cuerda para saltar", 300);  
AltaArticulo("Tobillos con peso", 850);  
AltaArticulo("Set de dardos", 400);  
AltaArticulo("Bate de béisbol", 1900);  
AltaArticulo("Bola de voleibol", 850);  
AltaArticulo("Aro de baloncesto", 2700);  
AltaArticulo("Silla de camping", 1100);  
AltaArticulo("Tienda de campaña", 8700);  
AltaArticulo("Colchoneta de yoga", 1200);  
AltaArticulo("Barra de dominadas", 1900);  
AltaArticulo("Reloj deportivo", 6500);  
AltaArticulo("Monopatín eléctrico", 18000);  
AltaArticulo("Kit de pesca", 3200);  
AltaArticulo("Bolsa de golf", 7600);  
AltaArticulo("Raqueta de bádminton", 1600);  
AltaArticulo("Patineta longboard", 3300);  
AltaArticulo("Bola de rugby", 1050);  
AltaArticulo("Kit de snorkel", 1800);  
AltaArticulo("Camiseta de compresión", 1300);  
AltaArticulo("Gorra deportiva", 400);  
AltaArticulo("Balón medicinal", 2000);  
AltaArticulo("Kit de arquería", 9800);  
AltaArticulo("Soga de escalada", 5600);  
AltaArticulo("Casco de esquí", 3700);  
AltaArticulo("Gafas de ciclismo", 900);
```

Esta lista incluye una variedad de artículos deportivos y de actividad física, con precios que corresponden a valores típicos de mercado.



Para las ventas y las subastas



Aquí tienes 8 ventas y 8 subastas siguiendo el mismo formato que proporcionaste:

Ventas

csharp

Copy code

```
AltaVenta("Caminata en el bosque", "ABIERTA", DateTime.ParseExact("12/02/2024", "dd/MM/
AltaVenta("Paseo en bicicleta", "ABIERTA", DateTime.ParseExact("15/03/2024", "dd/MM/yyy
AltaVenta("Clase de yoga", "ABIERTA", DateTime.ParseExact("22/04/2024", "dd/MM/yyyy", r
AltaVenta("Día de spa", "ABIERTA", DateTime.ParseExact("30/05/2024", "dd/MM/yyyy", null
AltaVenta("Concierto al aire libre", "ABIERTA", DateTime.ParseExact("01/08/2024", "dd/M
AltaVenta("Cata de vinos", "ABIERTA", DateTime.ParseExact("10/09/2024", "dd/MM/yyyy", r
AltaVenta("Taller de pintura", "ABIERTA", DateTime.ParseExact("15/10/2024", "dd/MM/yyyy
AltaVenta("Excursión a la montaña", "ABIERTA", DateTime.ParseExact("25/11/2024", "dd/MM/
```

Subastas

csharp

 Copy code

```
AltaSubasta("Torneo de ajedrez", "ABIERTA", DateTime.ParseExact("12/03/2024", "dd/MM/yy
AltaSubasta("Subasta de arte", "ABIERTA", DateTime.ParseExact("20/04/2024", "dd/MM/yyyy
AltaSubasta("Rally de coches", "ABIERTA", DateTime.ParseExact("01/06/2024", "dd/MM/yyyy
AltaSubasta("Subasta de antigüedades", "ABIERTA", DateTime.ParseExact("15/07/2024", "dd
AltaSubasta("Concurso de cocina", "ABIERTA", DateTime.ParseExact("05/08/2024", "dd/MM/y
AltaSubasta("Maratón de lectura", "ABIERTA", DateTime.ParseExact("12/09/2024", "dd/MM/y
AltaSubasta("Competencia de fotografía", "ABIERTA", DateTime.ParseExact("30/10/2024", "dd
AltaSubasta("Fiesta de disfraces", "ABIERTA", DateTime.ParseExact("15/11/2024", "dd/MM/
```

Si necesitas más ajustes o variaciones, házmelo saber.



Message ChatGPT



PROGRAM.CS

```
using LogicaNegocio;

using System;

namespace InterfazUsuario
{
    internal class Program
    {
        static void Main(string[] args)
        {
            MenuTipoUsuario();

        }

        // Crear una instancia de la clase Sistema
        private static Sistema miSistema = new Sistema();

        /// <summary>
        /// La estructura modular de los menus permite navegarlos de distinta
        /// forma dependiendo del rol que tengas asignado (el primer menu lo determina).
        /// Cada vez que se selecciona una opción se ejecuta una funcion o se redirige a otro menu.
        /// Cada vez que se ejecuta un menu se borra el contenido en pantalla.
        /// La función MostrarOpcionesMenu(string[] opciones) permite imprimir las opciones de los
        menus en pantalla.
        /// </summary>

        #region Menu

        #region Utilidades

        static void MostrarOpcionesMenu(string[] opciones)
        {
            Console.WriteLine("-----");
```

```

        Console.WriteLine(opciones[0]);

        for (int i = 1; i < opciones.Length; i++)
        {
            Console.WriteLine(opciones[i]);
        }

        Console.WriteLine("-----");
    }

    static void MostrarOpcionesMenuPorTipoUsuario(string tipoUsuario, string[] opcionesCliente,
string[] opcionesAdministrador, string[] opcionesTester)
    {
        switch (tipoUsuario)
        {
            case "CLIENTE":
                MostrarOpcionesMenu(opcionesCliente);

                break;

            case "ADMINISTRADOR":
                MostrarOpcionesMenu(opcionesAdministrador);

                break;

            case "TESTER":
                MostrarOpcionesMenu(opcionesTester);

                break;
        }
    }

    static void VolverAlMenu()
    {
        Console.WriteLine("Precione Intro para volver al menu");

        Console.ReadLine();
    }

    /// <summary>

    /// Implementa el bloque try catch a los menus

```

```

/// idMenu = 0 == MenuArticulo

/// idMenu = 1 == MenuBuscarArticulo

/// idMenu = 2 == MenuPublicacion

/// idMenu = 3 == MenuMostrarPublicacion

/// idMenu = 4 == MenuBuscarPublicacion

/// idMenu = 5 == MenuOfetaSubasta

/// idMenu = 6 == MenuAltaPublicacion

/// idMenu = 7 == MenuUsuario

/// idMenu = 8 == MenuMostrarUsuario

/// idMenu = 9 == MenuBuscarUsuario

/// idMenu = 10 == MenuAltaUsuario

/// </summary>

static bool ValidacionMenu(int idMenu, string tipoUsuario)
{
    try
    {
        switch (idMenu)
        {
            case 0:
                MenuArticulo(tipoUsuario);

                break;

            case 1:
                MenuBuscarArticulo(tipoUsuario);

                break;

            case 2:
                MenuPublicacion(tipoUsuario);

                break;

            case 3:
                MenuMostrarPublicacion(tipoUsuario);

```

```
        break;

    case 4:

        MenuBuscarPublicacion(tipoUsuario);

        break;

    case 5:

        MenuOfetaSubasta(tipoUsuario);

        break;

    case 6:

        MenuAltaPublicacion(tipoUsuario);

        break;

    case 7:

        MenuUsuario(tipoUsuario);

        break;

    case 8:

        MenuMostrarUsuario(tipoUsuario);

        break;

    case 9:

        MenuBuscarUsuario(tipoUsuario);

        break;

    case 10:

        MenuAltaUsuario(tipoUsuario);

        break;

    }

}

catch (InvalidOperationException ex)

{

    Console.WriteLine($"Error de operación: {ex.Message}");

    VolverAlMenu();

    return false;

}
```

```

    }

    catch (ArgumentNullException ex)
    {
        Console.WriteLine($"Falta un argumento obligatorio: {ex.Message}");

        VolverAlMenu();

        return false;
    }

    catch (ArgumentException ex)
    {
        Console.WriteLine($"Argumento inválido: {ex.Message}");

        VolverAlMenu();

        return false;
    }

    catch (Exception ex)
    {
        Console.WriteLine($"Error inesperado: {ex.Message}");

        VolverAlMenu();

        return false;
    }

    return true;
}

#endregion

#region Principal

static void MenuTipoUsuario()
{
    bool valido = false;

    string[] opciones = new string[] { "Menu selección tipo de usuario", "S. Salir del sistema", "1.
Usuario", "2. Administrador", "3. Tester" };

    string tipoUsuario;

```

```
while (!valido)
{
    Console.Clear();

    MostrarOpcionesMenu(opciones); // Imprime las opciones del array opciones

    char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

    switch (opcionSeleccionada)
    {
        case 'S':
            valido = true;

            break;

        case 's':
            valido = true;

            break;

        case '1':
            valido = true;

            tipoUsuario = "CLIENTE";

            Menu(tipoUsuario);

            break;

        case '2':
            valido = true;

            tipoUsuario = "ADMINISTRADOR";

            Menu(tipoUsuario);

            break;

        case '3':
            valido = true;

            tipoUsuario = "TESTER";

            Menu(tipoUsuario);

            break;
    }
}
```



```

    }
}

static void Menu(string tipoUsuario)
{
    bool valido = false;

    string[] opciones = new string[] { "Menu", "0. ...", "1. Artículos", "2. Publicaciones", "3. Usuarios" };

    while (!valido)
    {
        if (tipoUsuario == "CLIENTE" || tipoUsuario == "ADMINISTRADOR" || tipoUsuario == "TESTER")
        {
            Console.Clear();

            MostrarOpcionesMenu(opciones); // Imprime las opciones del array opciones

            char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

            switch (opcionSeleccionada)
            {
                case '0':
                    valido = true;

                    MenuTipoUsuario();

                    break;

                case '1':
                    valido = ValidacionMenu(0, tipoUsuario);

                    break;

                case '2':
                    valido = ValidacionMenu(2, tipoUsuario);

                    break;

                case '3':

```

```

        valido = ValidacionMenu(7, tipoUsuario);

        break;
    }
}
}
}

#endregion

#region Artículo

static void MenuArticulo(string tipoUsuario)
{
    bool valido = false;

    string[] opcionesCliente = new string[] { "Menu Articulo", "0. ...", "1. Mostrar catálogo", "2. Buscar"
};

    string[] opcionesAdministrador = new string[] { "Menu Articulo", "0. ...", "1. Mostrar catálogo", "2.
Buscar", "3. Dar de alta articulo" };

    while (!valido)
    {
        Console.Clear();

        MostrarOpcionesMenuPorTipoUsuario(tipoUsuario, opcionesCliente, opcionesAdministrador,
opcionesAdministrador); // Imprime las opciones del menu por tipo de usuario

        char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

        valido = ProcesamientoOpcionArticulo(tipoUsuario, opcionSeleccionada, valido);
    }
}

static void MenuBuscarArticulo(string tipoUsuario)
{
    bool valido = false;

    string[] opciones = new string[] { "Menu Buscar", "0. ...", "1. Buscar artículo por ID", "2. Buscar
artículo por Nombre", "3. Buscar artículo por categoría" };

```

```

while (!valido)
{
    Console.Clear();

    MostrarOpcionesMenu(opciones); // Imprime las opciones del menu

    char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

    valido = OpcionBuscarArticulo(tipoUsuario, opcionSeleccionada, valido);
}
}

#endregion

#region Publicacion

static void MenuPublicacion(string tipoUsuario)
{
    bool valido = false;

    string[] opcionesCliente = new string[] { "Menu Publicacion", "0. ...", "1. Mostrar", "2. Buscar", "3. Ofertar" };

    string[] opcionesAdministrador = new string[] { "Menu Publicacion", "0. ...", "1. Mostrar", "2. Buscar", "3. Ofertar", "4. Dar de alta" };

    while (!valido)
    {
        Console.Clear();

        MostrarOpcionesMenuPorTipoUsuario(tipoUsuario, opcionesCliente, opcionesAdministrador, opcionesAdministrador); // Imprime las opciones del menu por tipo de usuario

        char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

        valido = ProcesamientoOpcionPublicacion(tipoUsuario, opcionSeleccionada, valido);
    }
}
}

```

```

static void MenuMostrarPublicacion(string tipoUsuario)
{
    bool valido = false;

    string[] opciones = new string[] { "Menu Mostrar", "0. ...", "1. Mostrar todas las publicaciones", "2.
Mostrar todas las ventas", "3. Mostrar todas las subastas" };

    while (!valido)
    {
        Console.Clear();

        MostrarOpcionesMenu(opciones); // Imprime las opciones del menu

        char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

        valido = OpcionMostrarPublicacion(tipoUsuario, opcionSeleccionada, valido);
    }
}

static void MenuBuscarPublicacion(string tipoUsuario)
{
    bool valido = false;

    string[] opciones = new string[] { "Menu Búsqueda", "0. ...", "1. Buscar publicaciones por ID", "2.
Buscar publicaciones por Nombre", "3. Buscar publicación por fecha" };

    while (!valido)
    {
        Console.Clear();

        MostrarOpcionesMenu(opciones); // Imprime las opciones del menu

        char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

        valido = OpcionBuscarPublicacion(tipoUsuario, opcionSeleccionada, valido);
    }
}

```

```

static void MenuOfetaSubasta(string tipoUsuario)
{
    bool valido = false;

    string[] opciones = new string[] { "Menu Ofertar", "0. ...", "1. Ofertar con tú ID", "2. Ofertar con tú
Nombre" };

    while (!valido)
    {
        Console.Clear();

        MostrarOpcionesMenu(opciones); // Imprime las opciones del menu

        char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

        valido = OpcionOfertaSubasta(tipoUsuario, opcionSeleccionada, valido);
    }
}

static void MenuAltaPublicacion(string tipoUsuario)
{
    bool valido = false;

    string[] opcionesCliente = new string[] { "" };

    string[] opcionesAdministrador = new string[] { "Menu Alta", "0. ...", "1. Dar de alta venta", "2. Dar
de alta subasta" };

    string[] opcionesTester = new string[] { "Menu Alta", "0. ...", "1. Dar de alta publicacion", "2. Dar de
alta venta", "3. Dar de alta subasta" };

    while (!valido)
    {
        Console.Clear();

        MostrarOpcionesMenuPorTipoUsuario(tipoUsuario, opcionesCliente, opcionesAdministrador,
opcionesTester); // Imprime las opciones del menu por tipo de usuario

        char.TryParse(Console.ReadLine(), out char opcionSeleccionada);
    }
}

```

```

        valido = ProcesamientoOpcionAltaPublicacion(tipoUsuario, opcionSeleccionada, valido);
    }
}

#endregion

#region Usuario

static void MenuUsuario(string tipoUsuario)
{
    bool valido = false;

    string[] opcionesCliente = new string[] { "Menu Usuarios", "0. ...", "1. Mostrar todos los clientes",
"2. Buscar" };

    string[] opcionesAdministrador = new string[] { "Menu Usuarios", "0. ...", "1. Mostrar", "2. Buscar",
"3. Dar de alta" };

    while (!valido)
    {
        Console.Clear();

        MostrarOpcionesMenuPorTipoUsuario(tipoUsuario, opcionesCliente, opcionesAdministrador,
opcionesAdministrador); // Imprime las opciones del menu por tipo de usuario

        char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

        valido = ProcesamientoOpcionUsuario(tipoUsuario, opcionSeleccionada, valido);
    }
}

static void MenuMostrarUsuario(string tipoUsuario)
{
    bool valido = false;

    string[] opciones = new string[] { "Menu Mostrar", "0. ...", "1. Mostrar todos los usuarios", "2.
Mostrar todos los clientes", "3. Mostrar todos los administradores" };

    while (!valido)
    {

```

```

        Console.Clear();

        MostrarOpcionesMenu(opciones); // Imprime las opciones del menu

        char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

        valido = OpcionMostrarUsuario(tipoUsuario, opcionSeleccionada, valido);
    }
}

static void MenuBuscarUsuario(string tipoUsuario)
{
    bool valido = false;

    string[] opciones = new string[] { "Menu Busqueda", "0. ...", "1. Buscar usuario por ID", "2. Buscar
usuario por Nombre" };

    while (!valido)
    {
        Console.Clear();

        MostrarOpcionesMenu(opciones); // Imprime las opciones del menu

        char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

        valido = OpcionBuscarUsuario(tipoUsuario, opcionSeleccionada, valido);
    }
}

static void MenuAltaUsuario(string tipoUsuario)
{
    bool valido = false;

    string[] opcionesCliente = new string[] { "" };

    string[] opcionesAdministrador = new string[] { "Menu Alta", "0. ...", "1. Dar de alta cliente", "2. Dar
de alta administrador" };

    string[] opcionesTester = new string[] { "Menu Alta", "0. ...", "1. Dar de alta usuario", "2. Dar de alta
cliente", "3. Dar de alta administrador" };

```

```

while (!valido)
{
    Console.Clear();

    MostrarOpcionesMenuPorTipoUsuario(tipoUsuario, opcionesCliente, opcionesAdministrador,
opcionesTester); // Imprime las opciones del menu por tipo de usuario

    char.TryParse(Console.ReadLine(), out char opcionSeleccionada);

    valido = ProcesamientoOpcionAltaUsuario(tipoUsuario, opcionSeleccionada, valido);
}
}
#endregion
#endregion

/// <summary>
/// Las opciones del menu trabajan con la variable opcionSeleccionada,
/// esta variable es modificada si el menu es diferente para algún tipo de usuario
/// ya que el indice que se muestra para cada opción en el menu cambia.
/// Por ejemplo, si cliente tiene 2 opciones disponibles en el menu y administrador
/// tiene 3 opciones disponibles, con una función intermedia se modifica el input del cliente
/// si este es == 3 y se remplaza con 99.
/// Esto permite evitar que el usuario acceda a una opción que no tiene disponible.
/// </summary>
#region Opciones del menu
#region Articulo

static bool ProcesamientoOpcionArticulo(string tipoUsuario, int opcionSeleccionada, bool valido)
{
    if (tipoUsuario == "CLIENTE")
    {
        if (opcionSeleccionada == '3')

```



```

    {
        opcionSeleccionada = 'F';
    }
}

return OpcionArticulo(tipoUsuario, opcionSeleccionada, valido);
}

static bool OpcionArticulo(string tipoUsuario, int opcionSeleccionada, bool valido)
{

    switch (opcionSeleccionada)
    {
        case '0':
            valido = true;

            Menu(tipoUsuario);

            break;

        case '1':

            Console.WriteLine(new string("\n", 40));

            Console.Clear();

            ImprimirArticulo(miSistema.ObtenerArticulos(), true);

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(0, tipoUsuario);

            break;

        case '2':

            valido = ValidacionMenu(1, tipoUsuario);

            break;

        case '3':

            AltaArticulo();

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

```

```

        valido = ValidacionMenu(0, tipoUsuario);

        break;
    }

    return valido;
}

static bool OpcionBuscarArticulo(string tipoUsuario, int opcionSeleccionada, bool valido)
{
    switch (opcionSeleccionada)
    {
        case '0':

            valido = ValidacionMenu(0, tipoUsuario);

            break;

        case '1':

            ObtenerArticuloPorId();

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(1, tipoUsuario);

            break;

        case '2':

            ObtenerArticuloPorNombre();

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(1, tipoUsuario);

            break;

        case '3':

            ObtenerArticuloPorCategoria();

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(1, tipoUsuario);

            break;
    }

    return valido;
}

```

```

    }

    #endregion

    #region Publicacion

    static bool ProcesamientoOpcionPublicacion(string tipoUsuario, int opcionSeleccionada, bool
valido)
    {
        if (tipoUsuario == "CLIENTE")
        {
            if (opcionSeleccionada == '4')
            {
                opcionSeleccionada = 'F';
            }
        }
    }

    return OpcionPublicacion(tipoUsuario, opcionSeleccionada, valido);
}

static bool OpcionPublicacion(string tipoUsuario, int opcionSeleccionada, bool valido)
{
    // Ejecución de las opciones del menu por tipo de usuario
    switch (opcionSeleccionada)
    {
        case '0':
            valido = true;
            Menu(tipoUsuario);
            break;
        case '1':
            valido = ValidacionMenu(3, tipoUsuario);
            break;
        case '2':
            valido = ValidacionMenu(4, tipoUsuario);

```

```

        break;
    case '3':
        valido = ValidacionMenu(5, tipoUsuario);

        break;
    case '4':
        valido = ValidacionMenu(6, tipoUsuario);

        break;
    }

    return valido;
}

static bool OpcionMostrarPublicacion(string tipoUsuario, int opcionSeleccionada, bool valido)
{
    switch (opcionSeleccionada)
    {
        case '0':
            valido = ValidacionMenu(2, tipoUsuario);

            break;
        case '1':
            Console.WriteLine(new string("\n", 40));

            Console.Clear();

            ImprimirPublicacion(miSistema.ObtenerPublicaciones(false, false), true);

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(3, tipoUsuario);

            break;
        case '2':
            Console.WriteLine(new string("\n", 40));

            Console.Clear();

            ImprimirPublicacion(miSistema.ObtenerPublicaciones(true, false), true);

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

```

```

        valido = ValidacionMenu(3, tipoUsuario);

        break;
    case '3':

        Console.WriteLine(new string('\n', 40));

        Console.Clear();

        ImprimirPublicacion(miSistema.ObtenerPublicaciones(false, true), true);

        VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

        valido = ValidacionMenu(3, tipoUsuario);

        break;
    }

    return valido;
}

static bool OpcionBuscarPublicacion(string tipoUsuario, int opcionSeleccionada, bool valido)
{
    switch (opcionSeleccionada)
    {
        case '0':

            valido = ValidacionMenu(2, tipoUsuario);

            break;

        case '1':

            ObtenerPublicacionPorId();

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(4, tipoUsuario);

            break;

        case '2':

            ObtenerPublicacionPorNombre();

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(4, tipoUsuario);

            break;
    }
}

```

```

        case '3':

            ObtenerPublicacionPorFecha();

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(4, tipoUsuario);

            break;

    }

    return valido;

}

static bool OpcionOfertaSubasta(string tipoUsuario, int opcionSeleccionada, bool valido)
{
    switch (opcionSeleccionada)
    {
        case '0':

            valido = ValidacionMenu(2, tipoUsuario);

            break;

        case '1':

            AltaOfertaPorId();

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(5, tipoUsuario);

            break;

        case '2':

            AltaOfertaPorNombre();

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(5, tipoUsuario);

            break;

    }

    return valido;

}

static bool ProcesamientoOpcionAltaPublicacion(string tipoUsuario, int opcionSeleccionada, bool
valido)

```

```

{
    if (tipoUsuario == "ADMINISTRADOR")
    {
        if (opcionSeleccionada == '3')
        {
            opcionSeleccionada = 'F';
        }
        if (opcionSeleccionada == '2')
        {
            opcionSeleccionada = '3';
        }
        if (opcionSeleccionada == '1')
        {
            opcionSeleccionada = '2';
        }
    }

    return OpcionAltaPublicacion(tipoUsuario, opcionSeleccionada, valido);
}

static bool OpcionAltaPublicacion(string tipoUsuario, int opcionSelecccionada, bool valido)
{
    switch (opcionSelecccionada)
    {
        case '0':
            valido = ValidacionMenu(2, tipoUsuario);
            break;
        case '1':
            AltaPublicacion();
            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro
    }
}

```

```

        valido = ValidacionMenu(6, tipoUsuario);

        break;

    case '2':

        AltaVenta();

        VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

        valido = ValidacionMenu(6, tipoUsuario);

        break;

    case '3':

        AltaSubasta();

        VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

        valido = ValidacionMenu(6, tipoUsuario);

        break;

    }

    return valido;

}

#endregion

#region Usuario

static bool ProcesamientoOpcionUsuario(string tipoUsuario, int opcionSeleccionada, bool valido)
{
    if (tipoUsuario == "CLIENTE")
    {
        if (opcionSeleccionada == '3')
        {
            opcionSeleccionada = 'F';
        }

        if (opcionSeleccionada == '4')
        {
            opcionSeleccionada = 'F';
        }
    }
}

```



```

        if (opcionSeleccionada == '1')
        {
            opcionSeleccionada = '4';
        }
    }
    else
    {
        if (opcionSeleccionada == '4')
        {
            opcionSeleccionada = 'F';
        }
    }

    return OpcionUsuario(tipoUsuario, opcionSeleccionada, valido);
}

static bool OpcionUsuario(string tipoUsuario, int opcionSeleccionada, bool valido)
{
    switch (opcionSeleccionada)
    {
        case '0':
            valido = true;
            Menu(tipoUsuario);
            break;
        case '1':
            valido = ValidacionMenu(8, tipoUsuario);
            break;
        case '2':
            valido = ValidacionMenu(9, tipoUsuario);
            break;
    }
}

```

```

    case '3':

        valido = ValidacionMenu(10, tipoUsuario);

        break;

    case '4':

        Console.WriteLine(new string("\n", 40));

        Console.Clear();

        ImprimirUsuario(miSistema.ObtenerUsuarios(true, false));

        VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

        valido = ValidacionMenu(7, tipoUsuario);

        break;

    }

    return valido;

}

static bool OpcionMostrarUsuario(string tipoUsuario, int opcionSeleccionada, bool valido)
{
    switch (opcionSeleccionada)
    {
        case '0':

            valido = ValidacionMenu(7, tipoUsuario);

            break;

        case '1':

            Console.WriteLine(new string("\n", 40));

            Console.Clear();

            ImprimirUsuario(miSistema.ObtenerUsuarios(false, false));

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(8, tipoUsuario);

            break;

        case '2':

            Console.WriteLine(new string("\n", 40));

```

```

        Console.Clear();

        ImprimirUsuario(miSistema.ObtenerUsuarios(true, false));

        VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

        valido = ValidacionMenu(8, tipoUsuario);

        break;

    case '3':

        Console.WriteLine(new string('\n', 40));

        Console.Clear();

        ImprimirUsuario(miSistema.ObtenerUsuarios(false, true));

        VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

        valido = ValidacionMenu(8, tipoUsuario);

        break;

    }

    return valido;

}

static bool OpcionBuscarUsuario(string tipoUsuario, int opcionSeleccionada, bool valido)
{
    switch (opcionSeleccionada)
    {
        case '0':

            valido = ValidacionMenu(7, tipoUsuario);

            break;

        case '1':

            ObtenerUsuarioPorId();

            VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

            valido = ValidacionMenu(9, tipoUsuario);

            break;

        case '2':

            ObtenerUsuarioPorNombre();

```

```

        VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

        valido = ValidacionMenu(9, tipoUsuario);

        break;
    }

    return valido;
}

static bool ProcesamientoOpcionAltaUsuario(string tipoUsuario, int opcionSeleccionada, bool
valido)
{
    if (tipoUsuario == "ADMINISTRADOR")
    {
        if (opcionSeleccionada == '3')
        {
            opcionSeleccionada = 'F';
        }

        if (opcionSeleccionada == '2')
        {
            opcionSeleccionada = '3';
        }

        if (opcionSeleccionada == '1')
        {
            opcionSeleccionada = '2';
        }
    }
}

return OpcionAltaUsuario(tipoUsuario, opcionSeleccionada, valido);
}

static bool OpcionAltaUsuario(string tipoUsuario, int opcionSelecccionada, bool valido)
{
    switch (opcionSelecccionada)

```

```

{
    case '0':
        valido = ValidacionMenu(7, tipoUsuario);

        break;

    case '1':
        AltaUsuario();

        VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

        valido = ValidacionMenu(10, tipoUsuario);

        break;

    case '2':
        AltaCliente();

        VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

        valido = ValidacionMenu(10, tipoUsuario);

        break;

    case '3':
        AltaAdministrador();

        VolverAlMenu(); // Limpia la consola cuando el usuario preciona Intro

        valido = ValidacionMenu(10, tipoUsuario);

        break;

}

return valido;

}

#endregion

#endregion

```

/// <summary>

/// Las funciones de impresión son las menores posibles para evitar diferencias en la impresión.

/// Estas imprimen los datos basandose en listas de datos.

/// Tambien tienen booleanos como margenesGrandes o vistaResumida que sirven para facilitar

```

/// la lectura de los datos por parte del usuario al utilizar el programa

/// </summary>

#region Impresion de listas

#region Articulo

static void ImprimirArticulo(List<Articulo> articulos, bool margenesGrandes)
{
    int margenes = 35;

    string menuAnidado = " ";

    if (margenesGrandes)
    {
        margenes = 70;

        menuAnidado = "";
    }

    for (int i = 0; i < articulos.Count; i++)
    {
        Console.WriteLine(new string('-', margenes));

        // Mostramos los detalles del Artículo

        Console.WriteLine($"{menuAnidado}ID: {articulos[i].Id}");

        Console.WriteLine($"{menuAnidado}Nombre: {articulos[i].Nombre}");

        Console.WriteLine($"{menuAnidado}Precio: {articulos[i].Precio}");

        Console.WriteLine($"{menuAnidado}Categoría: {articulos[i].Categoria}");

    }

    Console.WriteLine(new string('-', margenes));

}

#endregion

#region Publicacion

static void ImprimirPublicacion(List<Publicacion> publicaciones, bool vistaResumida)
{

```

```

for (int i = 0; i < publicaciones.Count; i++)
{
    Console.WriteLine(new string('-', 70));

    // Mostramos los detalles de las publicaciones

    Console.WriteLine($"ID: {publicaciones[i].Id}");

    Console.WriteLine($"Nombre: {publicaciones[i].Nombre}");

    Console.WriteLine($"Estado: {publicaciones[i].Estado}");

    Console.WriteLine($"Fecha de publicación: {publicaciones[i].Fecha}");

    Console.WriteLine($"Articulos: {publicaciones[i].Articulos.Count}");

    if (publicaciones[i].Articulos.Count > 0) // Si hay articulos imprimimos sus ids
    {
        Console.WriteLine($"Id de los articulos asociados:
{miSistema.ParseoArticulo(publicaciones[i].Articulos)}");

        if (!vistaResumida) // Si no es vista resumida mostramos los datos de los articulos
        {
            ImprimirArticulo(publicaciones[i].Articulos, false); // Imprime los datos de los articulos
asociados
        }
    }

    if (publicaciones[i] is Venta venta)
    {
        if (venta.OfertaRelampago)
        {
            Console.WriteLine($"Oferta relampago: Si");
        }
        else
        {
            Console.WriteLine($"Oferta relampago: No");
        }
    }
}

```

```
        Console.WriteLine($"Precio venta {miSistema.ConsultarPrecioVenta(publicaciones[i],  
publicaciones[i].Articulos)}");
```

```
    }
```

```
    if (publicaciones[i] is Subasta subasta)
```

```
    {
```

```
        Console.WriteLine($"Ofertas: {subasta.Ofertas.Count}");
```

```
        if (subasta.Ofertas.Count > 0) // Si hay ofertas imprimimos sus ids
```

```
        {
```

```
            Console.WriteLine($"Id de las ofertas asociadas:  
{miSistema.ParseoOferta(subasta.Ofertas)}");
```

```
            if (!vistaResumida) // Si no es vista resumida mostramos los datos de las ofertas
```

```
            {
```

```
                ImprimirOferta(subasta.Ofertas); // Imprime los datos de las ofertas asociadas
```

```
            }
```

```
        }
```

```
    }
```

```
    if (publicaciones[i].FechaFin != DateTime.MinValue)
```

```
    {
```

```
        Console.WriteLine($"Cliente: {publicaciones[i].Cliente}");
```

```
        Console.WriteLine($"Administrador: {publicaciones[i].Administrador}");
```

```
        Console.WriteLine($"Fecha Fin: {publicaciones[i].FechaFin}");
```

```
    }
```

```
}
```

```
Console.WriteLine(new string('-', 70));
```

```
}
```

```
#endregion
```

```
#region Usuario
```

```
static void ImprimirUsuario(List<Usuario> usuarios)
```

```
{
```



```

if (usuarios.Count > 0)
{
    for (int i = 0; i < usuarios.Count; i++)
    {
        Console.WriteLine(new string('-', 70));

        if (usuarios[i] is Administrador administrador)
        {
            // Mostramos los detalles del Administrador

            Console.WriteLine($"ID: {administrador.Id}");

            Console.WriteLine($"Nombre: {administrador.Nombre}");

            Console.WriteLine($"Apellido: {administrador.Apellido}");

            Console.WriteLine($"Email: {administrador.Email}");

        }

        if (usuarios[i] is Cliente cliente && miSistema.HayCliente())
        {
            // Mostramos los detalles del Usuario

            Console.WriteLine($"ID: {cliente.Id}");

            Console.WriteLine($"Nombre: {cliente.Nombre}");

            Console.WriteLine($"Apellido: {cliente.Apellido}");

            Console.WriteLine($"Email: {cliente.Email}");

            Console.WriteLine($"Saldo: {cliente.Saldo}");

        }

    }

    Console.WriteLine(new string('-', 70));

}

if (!miSistema.HayCliente())
{
    Console.WriteLine("No existen clientes actualmente");

}

```

```

}

#endregion

#region Oferta

static void ImprimirOferta(List<Oferta> ofertas)
{
    string menuAnidado = " ";

    for (int i = 0; i < ofertas.Count; i++)
    {
        Console.WriteLine(new string('-', 35));

        // Mostramos los detalles de las ofertas

        Console.WriteLine($"{menuAnidado}ID: {ofertas[i].Id}");

        if (ofertas[i].Usuario is Usuario usuario)
        {
            Console.WriteLine($"{menuAnidado}Id usuario: {usuario.Id}");

            Console.WriteLine($"{menuAnidado}Nombre usuario: {usuario.Nombre}");

            Console.WriteLine($"{menuAnidado}Apellido usuario: {usuario.Apellido}");

        }

        Console.WriteLine($"{menuAnidado}Monto: {ofertas[i].Monto}");

        Console.WriteLine($"{menuAnidado}Fecha: {ofertas[i].Fecha}");

    }

    Console.WriteLine(new string('-', 35));

}

#endregion

#endregion

```

/// <summary>

/// En las funciones de busqueda se solicitan datos simples como id o nombre

/// pero tambien se habilita la posibilidad de buscar varios ids o nombres al mismo tiempo.

/// Luego se imprimen en pantalla las busquedas realizadas utilizando estos datos con las

```

/// funciones de Obtener que se encuentran en Sistema

/// </summary>

#region Busqueda

#region Articulo

static void ObtenerArticuloPorId()

{

    Console.Clear();

    // Solicitud datos

    Console.WriteLine("Id de los articulos separados por ,:");

    string idsCrudos = Console.ReadLine() ?? string.Empty;

    if (string.IsNullOrEmpty(idsCrudos))

    {

        Console.WriteLine("El id no puede ser vacío");

        return;

    }

    List<int> ids = miSistema.ParseoInt(idsCrudos); // Convierte el input del usuario en una lista de
ids

    if (ids == null || ids.Count == 0)

    {

        Console.WriteLine("No se encontraron ids correspondientes a los ids proporcionados");

        return;

    }

    List<Articulo> articulos = miSistema.ObtenerArticuloPorId(ids);

    if (articulos == null || articulos.Count == 0)

    {

        Console.WriteLine("No se encontraron articulos correspondientes a los ids proporcionados");

        return;

    }

    Console.WriteLine(new string('\n', 40));

```

```

        Console.Clear();

        ImprimirArticulo(articulos, true);
    }

    static void ObtenerArticuloPorNombre()
    {
        Console.Clear();

        // Solicitud datos

        Console.WriteLine("Nombre de los articulos separados por ,:");

        string nombresCrudos = Console.ReadLine() ?? string.Empty;

        if (string.IsNullOrEmpty(nombresCrudos))
        {
            Console.WriteLine("El nombre no puede ser vacío");

            return;
        }

        List<string> nombres = miSistema.ParseoString(nombresCrudos); // Convierte el input del
        usuario en una lista de nombres

        if (nombres == null || nombres.Count == 0)
        {
            Console.WriteLine("No se encontraron nombres correspondientes a los nombres
            proporcionados");

            return;
        }

        List<Articulo> articulos = miSistema.ObtenerArticuloPorNombre(nombres);

        if (articulos == null || articulos.Count == 0)
        {
            Console.WriteLine("No se encontraron articulos correspondientes a los nombres
            proporcionados");

            return;
        }
    }

```

```

        Console.WriteLine(new string('\n', 40));

        Console.Clear();

        ImprimirArticulo(articulos, true);
    }

    static void ObtenerArticuloPorCategoria()
    {
        Console.Clear();

        // Solicitud datos

        Console.WriteLine("Categoria de los articulos separados por ,:");

        string categoriasCrudas = Console.ReadLine() ?? string.Empty;

        if (string.IsNullOrEmpty(categoriasCrudas))
        {
            Console.WriteLine("La categoria no puede ser vacía");

            return;
        }

        List<string> categorias = miSistema.ParseoString(categoriasCrudas); // Convierte el input del
        usuario en una lista de nombres

        if (categorias == null || categorias.Count == 0)
        {
            Console.WriteLine("No se encontraron categorias correspondientes a las categorias
            proporcionados");

            return;
        }

        List<Articulo> articulos = miSistema.ObtenerArticuloPorCategoria(categorias);

        if (articulos == null || articulos.Count == 0)
        {
            Console.WriteLine("No se encontraron articulos correspondientes a las categorias
            proporcionadas");

            return;
        }
    }

```

```

        Console.WriteLine(new string('\n', 40));

        Console.Clear();

        ImprimirArticulo(articulos, true);
    }
#endregion

#region Publicacion

static void ObtenerPublicacionPorId()
{
    Console.Clear();

    // Solicitud datos

    Console.WriteLine("Id de las publicaciones separadas por ,:");

    string idsCrudos = Console.ReadLine() ?? string.Empty;

    if (string.IsNullOrEmpty(idsCrudos))
    {
        Console.WriteLine("El id no puede ser vacío");

        return;
    }

    List<int> ids = miSistema.ParseoInt(idsCrudos); // Convierte el input del usuario en una lista de
ids

    if (ids == null || ids.Count == 0)
    {
        Console.WriteLine("No se encontraron ids correspondientes a los ids proporcionados");

        return;
    }

    List<Publicacion> publicaciones = miSistema.ObtenerPublicacionPorId(ids, false, false);

    if (publicaciones == null || publicaciones.Count == 0)
    {
        Console.WriteLine("No se encontraron publicaciones correspondientes a los ids
proporcionados");
    }
}

```

```

        return;
    }

    Console.WriteLine(new string('\n', 40));

    Console.Clear();

    ImprimirPublicacion(publicaciones, false);
}

static void ObtenerPublicacionPorNombre()
{
    Console.Clear();

    // Solicitud datos

    Console.WriteLine("Nombre de las publicaciones separadas por ,:");

    string nombresCrudos = Console.ReadLine() ?? string.Empty;

    if (string.IsNullOrEmpty(nombresCrudos))
    {
        Console.WriteLine("El nombre no puede ser vacío");

        return;
    }

    List<string> nombres = miSistema.ParseoString(nombresCrudos); // Convierte el input del
    usuario en una lista de nombres

    if (nombres == null || nombres.Count == 0)
    {
        Console.WriteLine("No se encontraron nombres correspondientes a los nombres
    proporcionados");

        return;
    }

    List<Publicacion> publicaciones = miSistema.ObtenerPublicacionPorNombre(nombres, false,
    false);

    if (publicaciones == null || publicaciones.Count == 0)
    {

```

```
        Console.WriteLine("No se encontraron publicaciones correspondientes a los nombres  
proporcionados");
```

```
        return;
```

```
    }
```

```
    Console.WriteLine(new string('\n', 40));
```

```
    Console.Clear();
```

```
    ImprimirPublicacion(publicaciones, false);
```

```
}
```

```
static void ObtenerPublicacionPorFecha()
```

```
{
```

```
    Console.Clear();
```

```
    // Solicitud datos
```

```
    Console.WriteLine("Fecha inicio dd/MM/yyyy:");
```

```
    string fechalInicioCruda = Console.ReadLine() ?? string.Empty;
```

```
    if (string.IsNullOrEmpty(fechalInicioCruda))
```

```
    {
```

```
        Console.WriteLine("La fecha de inicio no puede ser vacía");
```

```
        return;
```

```
    }
```

```
    if (!DateTime.TryParse(fechalInicioCruda, out DateTime fechalInicio))
```

```
    {
```

```
        Console.WriteLine("El formato de la fecha de inicio no es válido. Use dd/MM/yyyy.");
```

```
        return;
```

```
    }
```

```
    Console.WriteLine("Fecha fin dd/MM/yyyy:");
```

```
    string fechaFinCruda = Console.ReadLine() ?? string.Empty;
```

```
    if (string.IsNullOrEmpty(fechaFinCruda))
```

```
    {
```

```
        Console.WriteLine("La fecha de fin no puede ser vacía");
```



```

        return;
    }

    if (!DateTime.TryParse(fechaFinCruda, out DateTime fechaFin))
    {
        Console.WriteLine("El formato de la fecha de inicio no es válido. Use dd/MM/yyyy.");
        return;
    }

    List<Publicacion> publicaciones = miSistema.ObtenerPublicacionPorFecha(fechaInicio,
fechaFin, false, false);

    if (publicaciones == null || publicaciones.Count == 0)
    {
        Console.WriteLine("No se encontraron publicaciones correspondientes al intervalo
proporcionado");
        return;
    }

    Console.WriteLine(new string('\n', 40));

    Console.Clear();

    ImprimirPublicacion(publicaciones, false);
}

#endregion

#region Usuario

static void ObtenerUsuarioPorId()
{
    Console.Clear();

    // Solicitud datos

    Console.WriteLine("Id de los usuarios separados por ,:");

    string idsCrudos = Console.ReadLine() ?? string.Empty;

    if (string.IsNullOrEmpty(idsCrudos))
    {

```

```

        Console.WriteLine("El id no puede ser vacío");

        return;
    }

    List<int> ids = miSistema.ParseoInt(idsCrudos); // Convierte el input del usuario en una lista de
ids
    if (ids == null || ids.Count == 0)
    {
        Console.WriteLine("No se encontraron ids correspondientes a los ids proporcionados");

        return;
    }

    List<Usuario> usuarios = miSistema.ObtenerUsuarioPorId(ids, false, false);

    if (usuarios == null || usuarios.Count == 0)
    {
        Console.WriteLine("No se encontraron usuarios correspondientes a los ids proporcionados");

        return;
    }

    Console.WriteLine(new string('\n', 40));

    Console.Clear();

    ImprimirUsuario(usuarios);
}

static void ObtenerUsuarioPorNombre()
{
    Console.Clear();

    // Solicitud datos

    Console.WriteLine("Nombre de los usuarios separados por ,:");

    string nombresCrudos = Console.ReadLine() ?? string.Empty;

    if (string.IsNullOrEmpty(nombresCrudos))
    {
        Console.WriteLine("El nombre no puede ser vacío");
    }
}

```

```

        return;
    }

    List<string> nombres = miSistema.ParseoString(nombresCrudos); // Convierte el input del
    usuario en una lista de nombres

    if (nombres == null || nombres.Count == 0)
    {
        Console.WriteLine("No se encontraron nombres correspondientes a los nombres
    proporcionados");

        return;
    }

    List<Usuario> usuarios = miSistema.ObtenerUsuarioPorNombre(nombres, false, false);

    if (usuarios == null || usuarios.Count == 0)
    {
        Console.WriteLine("No se encontraron usuarios correspondientes a los nombres
    proporcionados");

        return;
    }

    Console.WriteLine(new string('\n', 40));

    Console.Clear();

    ImprimirUsuario(usuarios);
}

#endregion

#endregion

/// <summary>

/// En las funciones de alta se solicitan datos minimos para poder dar de alta lo solicitado.

/// Algunos de los datos como la fecha de creación se asignan automaticamente (fecha.Now)

/// </summary>

#region Altas

```

```

#region Artículo

static void AltaArticulo()
{
    Console.Clear();

    // Solicitud datos

    Console.WriteLine("Ingrese los datos del artículo");

    Console.WriteLine("Nombre:");

    // ?? operador de coalescencia nula
    // Si es null devuelve el valor de la derecha
    // Si no es null devuelve el valor de la izquierda
    string nombre = Console.ReadLine() ?? string.Empty;
    if (string.IsNullOrEmpty(nombre))
    {
        Console.WriteLine("El nombre no puede ser vacío");
        return;
    }

    Console.WriteLine("Precio:");

    decimal precio;
    if (!decimal.TryParse(Console.ReadLine(), out precio) || precio < 0)
    {
        Console.WriteLine("El precio debe ser un número entero positivo");
        return;
    }

    Console.WriteLine("Categoría:");

    string categoria = Console.ReadLine() ?? string.Empty;
    if (string.IsNullOrEmpty(categoria))
    {
        Console.WriteLine("La categoria no puede ser vacía");
        return;
    }
}

```

```

    }

    miSistema.AltaArticulo(nombre, precio, categoria);

    Console.WriteLine("El articulo se ha creado exitosamente");
}

#endregion

#region Publicacion

static void AltaPublicacion()
{
    // Valores por defecto

    string estado = "ABIERTA";

    DateTime fecha = DateTime.Now;

    Cliente? cliente = null;

    Administrador? administrador = null;

    DateTime fechaFin = DateTime.MinValue;

    Console.Clear();

    // Solicitud datos

    Console.WriteLine("Ingrese los datos que desea asociar a la publicacion");

    Console.WriteLine("Nombre:");

    string nombre = Console.ReadLine() ?? string.Empty;

    if (string.IsNullOrEmpty(nombre))
    {
        Console.WriteLine("El nombre no puede ser vacío");

        return;
    }

    Console.WriteLine("Id de los articulos separados por ,:");

    string idsCrudos = Console.ReadLine() ?? string.Empty;

    if (string.IsNullOrEmpty(idsCrudos))

```

```

    {
        Console.WriteLine("El id no puede ser vacío");
        return;
    }

    List<int> ids = miSistema.ParseoInt(idsCrudos); // Convierte el input del usuario en una lista de
ids

    if (ids == null || ids.Count == 0)
    {
        Console.WriteLine("No se proporcionaron IDs válidos");
        return;
    }

    List<Articulo> articulos = miSistema.ObtenerArticuloPorId(ids); // Obtiene la lista de
publicaciones con los ids

    if (articulos == null || articulos.Count == 0)
    {
        Console.WriteLine("No se encontraron artículos correspondientes a los IDs proporcionados");
        return;
    }

    miSistema.AltaPublicacion(nombre, estado, fecha, articulos, cliente, administrador, fechaFin);

    Console.WriteLine("La publicación se ha creado exitosamente");
}

static void AltaVenta()
{
    // Valores por defecto

    string estado = "ABIERTA";

    DateTime fecha = DateTime.Now;

    Cliente? cliente = null;

    Administrador? administrador = null;

    DateTime fechaFin = DateTime.MinValue;

```

```

Console.Clear();

// Solicitud datos

Console.WriteLine("Ingrese los datos que desea asociar a la venta");

Console.WriteLine("Nombre:");

string nombre = Console.ReadLine() ?? string.Empty;

if (string.IsNullOrEmpty(nombre))

{

    Console.WriteLine("El nombre no puede ser vacío");

    return;

}

Console.WriteLine("Id de los articulos separados por ,:");

string idsCrudos = Console.ReadLine() ?? string.Empty;

if (string.IsNullOrEmpty(idsCrudos))

{

    Console.WriteLine("El id no puede ser vacío");

    return;

}

List<int> ids = miSistema.ParseoInt(idsCrudos); // Convierte el input del usuario en una lista de
ids

if (ids == null || ids.Count == 0)

{

    Console.WriteLine("No se proporcionaron IDs válidos");

    return;

}

List<Articulo> articulos = miSistema.ObtenerArticuloPorId(ids); // Obtiene la lista de
publicaciones con los ids

if (articulos == null || articulos.Count == 0)

{

    Console.WriteLine("No se encontraron artículos correspondientes a los IDs proporcionados");

```

```

        return;
    }

    Console.WriteLine("Es oferta relampago?\n1. Si\n2. No");

    int esOferta;

    if (!int.TryParse(Console.ReadLine(), out esOferta))
    {
        Console.WriteLine("Su respuesta debe ser 1 o 2");

        return;
    }

    bool ofertaRelampago = false;

    if ( esOferta == 1 )
    {
        ofertaRelampago = true;
    }

    miSistema.AltaVenta(nombre, estado, fecha, articulos, cliente, administrador, fechaFin,
ofertaRelampago);

    Console.WriteLine("La venta se ha creado exitosamente");
}

static void AltaSubasta()
{
    // Valores por defecto

    string estado = "ABIERTA";

    DateTime fecha = DateTime.Now;

    Cliente? cliente = null;

    Administrador? administrador = null;

    DateTime fechaFin = DateTime.MinValue;

    List<Oferta> ofertas = new List<Oferta>();

```



```

Console.Clear();

// Solicitud datos

Console.WriteLine("Ingrese los datos que desea asociar a la subasta");

Console.WriteLine("Nombre:");

string nombre = Console.ReadLine() ?? string.Empty;

if (string.IsNullOrEmpty(nombre))
{
    Console.WriteLine("El nombre no puede ser vacío");

    return;
}

Console.WriteLine("Id de los articulos separados por ,:");

string idsCrudos = Console.ReadLine() ?? string.Empty;

if (string.IsNullOrEmpty(idsCrudos))
{
    Console.WriteLine("El id no puede ser vacío");

    return;
}

List<int> ids = miSistema.ParseoInt(idsCrudos); // Convierte el input del usuario en una lista de
ids

if (ids == null || ids.Count == 0)
{
    Console.WriteLine("No se proporcionaron IDs válidos");

    return;
}

List<Articulo> articulos = miSistema.ObtenerArticuloPorId(ids); // Obtiene la lista de
publicaciones con los ids

if (articulos == null || articulos.Count == 0)
{
    Console.WriteLine("No se encontraron artículos correspondientes a los IDs proporcionados");

    return;
}

```

```
}
```

```
miSistema.AltaSubasta(nombre, estado, fecha, articulos, cliente, administrador, fechaFin,  
ofertas);
```

```
    Console.WriteLine("La subasta se ha creado exitosamente");
```

```
}
```

```
#endregion
```

```
#region Usuario
```

```
static void AltaUsuario()
```

```
{
```

```
    Console.Clear();
```

```
    // Solicitud datos
```

```
    Console.WriteLine("Ingrese los datos que desea asociar al usuario");
```

```
    Console.WriteLine("Nombre:");
```

```
    string nombre = Console.ReadLine() ?? string.Empty;
```

```
    if (string.IsNullOrEmpty(nombre))
```

```
    {
```

```
        Console.WriteLine("El nombre no puede ser vacío");
```

```
        return;
```

```
    }
```

```
    Console.WriteLine("Apellido:");
```

```
    string apellido = Console.ReadLine() ?? string.Empty;
```

```
    if (string.IsNullOrEmpty(apellido))
```

```
    {
```

```
        Console.WriteLine("El apellido no puede ser vacío");
```

```
        return;
```

```
    }
```

```
    Console.WriteLine("Email:");
```

```
    string email = Console.ReadLine() ?? string.Empty;
```

```
    if (string.IsNullOrEmpty(email))
```

```

{
    Console.WriteLine("El email no puede ser vacío");
    return;
}

if (email.IndexOf('@') == -1)
{
    Console.WriteLine("El email debe pertenecer a un domino (debe tener @)");
    return;
}

Console.WriteLine("Contraseña:");
string contrasenia = Console.ReadLine() ?? string.Empty;
if (string.IsNullOrEmpty(contrasenia))
{
    Console.WriteLine("La contraseña no puede ser vacía");
    return;
}

miSistema.AltaUsuario(nombre, apellido, email, contrasenia);
Console.WriteLine("El usuario se ha creado exitosamente");
}

static void AltaCliente()
{
    // Valores por defecto
    int saldo = 0;

    Console.Clear();

    // Solicitud datos
    Console.WriteLine("Ingrese los datos que desea asociar al cliente");
    Console.WriteLine("Nombre:");

```

```
string nombre = Console.ReadLine() ?? string.Empty;

if (string.IsNullOrEmpty(nombre))
{
    Console.WriteLine("El nombre no puede ser vacío");
    return;
}

Console.WriteLine("Apellido:");

string apellido = Console.ReadLine() ?? string.Empty;

if (string.IsNullOrEmpty(apellido))
{
    Console.WriteLine("El apellido no puede ser vacío");
    return;
}

Console.WriteLine("Email:");

string email = Console.ReadLine() ?? string.Empty;

if (string.IsNullOrEmpty(email))
{
    Console.WriteLine("El email no puede ser vacío");
    return;
}

if (email.IndexOf('@') == -1)
{
    Console.WriteLine("El email debe pertenecer a un domino (debe tener @)");
    return;
}

Console.WriteLine("Contraseña:");

string contrasenia = Console.ReadLine() ?? string.Empty;

if (string.IsNullOrEmpty(contrasenia))
{

```

```

        Console.WriteLine("La contraseña no puede ser vacía");
        return;
    }

    miSistema.AltaCliente(nombre, apellido, email, contrasenia, saldo);
    Console.WriteLine("El cliente se ha creado exitosamente");
}

static void AltaAdministrador()
{
    Console.Clear();

    // Solicitud datos
    Console.WriteLine("Ingrese los datos que desea asociar al administrador");
    Console.WriteLine("Nombre:");

    string nombre = Console.ReadLine() ?? string.Empty;
    if (string.IsNullOrEmpty(nombre))
    {
        Console.WriteLine("El nombre no puede ser vacío");
        return;
    }

    Console.WriteLine("Apellido:");
    string apellido = Console.ReadLine() ?? string.Empty;
    if (string.IsNullOrEmpty(apellido))
    {
        Console.WriteLine("El apellido no puede ser vacío");
        return;
    }

    Console.WriteLine("Email:");
    string email = Console.ReadLine() ?? string.Empty;
    if (string.IsNullOrEmpty(email))

```

```

{
    Console.WriteLine("El email no puede ser vacío");
    return;
}

if (email.IndexOf('@') == -1)
{
    Console.WriteLine("El email debe pertenecer a un domino (debe tener @)");
    return;
}

Console.WriteLine("Contraseña:");
string contrasenia = Console.ReadLine() ?? string.Empty;
if (string.IsNullOrEmpty(contrasenia))
{
    Console.WriteLine("La contraseña no puede ser vacía");
    return;
}

miSistema.AltaAdministrador(nombre, apellido, email, contrasenia);
Console.WriteLine("El administrador se ha creado exitosamente");
}

#endregion

#region Oferta

static void AltaOfertaPorId()
{
    // Valores por defecto
    DateTime fecha = DateTime.Now;

    Console.Clear();

    // Solicitud datos

```

```
Console.WriteLine("Ingrese los datos que desea asociar a la oferta");

Console.WriteLine("Id del comprador:");

int idUsuario;

if (!int.TryParse(Console.ReadLine(), out idUsuario) || idUsuario < 0)

{

    Console.WriteLine("El ID del usuario debe ser un número entero positivo");

    return;

}

Usuario? usuario = miSistema.ObtenerUsuarioPorId(idUsuario, true, false);

if (usuario == null)

{

    Console.WriteLine("No se encontró un usuario con ese nombre");

    return;

}

Console.WriteLine("Id de la subasta:");

int idPublicacion;

if (!int.TryParse(Console.ReadLine(), out idPublicacion) || idPublicacion < 0)

{

    Console.WriteLine("El ID de la subasta debe ser un número entero positivo");

    return;

}

Publicacion? publicacion = miSistema.ObtenerPublicacionPorId(idPublicacion, false, true);

if (publicacion == null)

{

    Console.WriteLine("No se encontró una subasta con ese ID");

    return;

}

Console.WriteLine("Monto a ofertar:");

decimal monto;
```

```

if (!decimal.TryParse(Console.ReadLine(), out monto) || monto <= 0)
{
    Console.WriteLine("El monto debe ser un número decimal positivo");
    return;
}

miSistema.AltaOferta(usuario, publicacion, monto, fecha);

Console.WriteLine("La oferta se ha realizado con éxito");
}

static void AltaOfertaPorNombre()
{
    // Valores por defecto
    DateTime fecha = DateTime.Now;

    Console.Clear();

    // Solicitud datos
    Console.WriteLine("Ingrese los datos que desea asociar a la oferta");
    Console.WriteLine("Nombre del comprador:");
    string nombre = Console.ReadLine() ?? string.Empty;
    if (string.IsNullOrEmpty(nombre))
    {
        Console.WriteLine("El nombre no puede ser vacío");
        return;
    }

    Usuario? usuario = miSistema.ObtenerUsuarioPorNombre(nombre, true, false);
    if (usuario == null)
    {
        Console.WriteLine("No se encontró un usuario con ese nombre");
        return;
    }
}

```



```

    }

    Console.WriteLine("Id de la subasta:");

    int idPublicacion;

    if (!int.TryParse(Console.ReadLine(), out idPublicacion) || idPublicacion < 0)
    {
        Console.WriteLine("El ID de la subasta debe ser un número entero positivo");

        return;
    }

    Publicacion? publicacion = miSistema.ObtenerPublicacionPorId(idPublicacion, false, true);

    if (publicacion == null)
    {
        Console.WriteLine("No se encontró una subasta con ese ID");

        return;
    }

    Console.WriteLine("Monto a ofertar:");

    decimal monto;

    if (!decimal.TryParse(Console.ReadLine(), out monto) || monto <= 0)
    {
        Console.WriteLine("El monto debe ser un número decimal positivo");

        return;
    }

    miSistema.AltaOferta(usuario, publicacion, monto, fecha);

    Console.WriteLine("La oferta se ha realizado con éxito");

}

#endregion

#endregion
}

```

ADMINISTRADOR.CS

```
using LogicaNegocio.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LogicaNegocio
{
    public class Administrador : Usuario
    {
        #region Constructor

        public Administrador(string nombre, string apellido, string email, string contrasenia)
            : base(nombre, apellido, email, contrasenia) // Llamada al constructor de la clase base (Usuario)
        {
        }

        #endregion

        #region Validación

        // Validación de Administrador, hereda de Usuario

        public override void Validar()
        {
        }

        #endregion

        #region Método Equals

        // Sobre escritura del metodo Equals que es usado por Contains

        public override bool Equals(object? obj)
        {
            if (obj != null && obj is Usuario)
```

```
{  
    Usuario usuario = (Usuario)obj;  
    return Nombre == usuario.Nombre && Apellido == usuario.Apellido;  
}  
    return false;  
}  
#endregion  
}  
}
```

ARTICULO.CS

```
using LogicaNegocio.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;


namespace LogicaNegocio
{
    public class Articulo : IValidate
    {
        #region Atributos de la clase

        private int _id;

        private static int s_ultId = 0; // Inicializado con el id siguiente a la ultima precarga

        private string _nombre = string.Empty; // Inicializado con una cadena vacía

        private decimal _precio = 0; // Inicializado con 0

        private string _categoria = string.Empty; // Inicializado con una cadena vacía

        #endregion


        #region Propiedades

        public int Id
        {
            get { return _id; } // Solo lectura, asignado internamente.
        }

        public string Nombre
        {
            get { return _nombre; }

            set { _nombre = EvaluarNombre(value); }
        }
    }
}
```

```

    }

    public decimal Precio
    {
        get { return _precio; }

        set { _precio = EvaluarPrecio(value); }
    }

    public string Categoria
    {
        get { return _categoria; }

        set { _categoria = EvaluarCategoria(value); }
    }
}

#endregion

#region Constructor

public Artículo(string nombre, decimal precio, string categoria)
{
    _id = Artículo.s_ultId; // Asigna el ID único
    Artículo.s_ultId++; // Incrementa el ID único
    Nombre = nombre;
    Precio = precio;
    Categoria = categoria;
}

#endregion

#region Validación

// Evaluaciones

private static string EvaluarNombre(string nombre)
{
    if (string.IsNullOrEmpty(nombre))

```

```
{
    throw new ArgumentException("El nombre no puede ser vacío");
}

return nombre;
}

private static decimal EvaluarPrecio(decimal precio)
{
    if (precio < 0)
    {
        throw new InvalidOperationException("El precio no puede ser negativo");
    }

    return precio;
}

private static string EvaluarCategoria(string categoria)
{
    if (string.IsNullOrEmpty(categoria))
    {
        throw new ArgumentException("La categoria no puede ser vacía");
    }

    return categoria;
}
```

// Validación de Artículo

```
public void Validar()
```

```
{
}
```

```
#endregion
```

```
#region Método Equals
```

```
// Sobre escritura del metodo Equals que es usado por Contains

public override bool Equals(object? obj)
{
    if (obj != null && obj is Artículo)
    {
        Artículo articulo = (Artículo)obj;
        return Nombre == articulo.Nombre;
    }
    return false;
}

#endregion
}
```

CLIENTE.CS

```
using LogicaNegocio.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;


namespace LogicaNegocio
{
    public class Cliente : Usuario, IValidate
    {
        #region Atributos de la clase

        private decimal _saldo = 0; // Inicializado en 0

        #endregion


        #region Propiedades

        public decimal Saldo
        {
            get { return _saldo; }

            set { _saldo = EvaluarSaldo(value); }
        }

        #endregion


        #region Constructor

        public Cliente(string nombre, string apellido, string email, string contrasenia, decimal saldo)
            : base(nombre, apellido, email, contrasenia) // Llamada al constructor de la clase base (Usuario)
        {
            Saldo = saldo;
        }
    }
}
```



```

}

#endregion

#region Validación

// Evaluaciones

private static decimal EvaluarSaldo(decimal saldo)
{
    if (saldo < 0)
    {
        throw new InvalidOperationException("El saldo no puede ser negativo");
    }

    return saldo;
}

// Validación de Cliente, hereda de Usuario

public override void Validar()
{
}

#endregion

#region Método Equals

// Sobre escritura del metodo Equals que es usado por Contains

public override bool Equals(object? obj)
{
    if (obj != null && obj is Usuario)
    {
        Usuario usuario = (Usuario)obj;

        return Nombre == usuario.Nombre && Apellido == usuario.Apellido;
    }
}

```

```
        return false;
    }
    #endregion
}
}
```

OFERTA.CS

```
using LogicaNegocio.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static System.Runtime.InteropServices.JavaScript.JSType;

namespace LogicaNegocio
{
    public class Oferta : IValidate
    {
        #region Atributos de la clase

        private int _id;

        private static int s_ultId = 0; // Inicializado con el id siguiente a la ultima precarga

        private Usuario? _usuario; // Inicializado con una instancia por defecto

        private decimal _monto = 0; // Inicializado con 0

        private DateTime _fecha = DateTime.Now; // Inicializado con la fecha actual

        #endregion

        #region Propiedades

        public int Id
        {
            get { return _id; } // Solo lectura, asignado internamente.
        }

        public Usuario? Usuario
        {
            get { return _usuario; }
```

```

        set { _usuario = EvaluarUsuario(value); }
    }

    public decimal Monto
    {
        get { return _monto; }

        set { _monto = EvaluarMonto(value); }
    }

    public DateTime Fecha
    {
        get { return _fecha; }

        set { _fecha = value; }
    }
}

#endregion

#region Constructor

public Oferta(Usuario? usuario, decimal monto, DateTime fecha)
{
    _id = Oferta.s_ultId; // Asigna el ID único
    Oferta.s_ultId++; // Incrementa el ID único
    Usuario = usuario;
    Monto = monto;
    Fecha = fecha;
}

#endregion

#region Validación

// Evaluaciones

private static Usuario EvaluarUsuario(Usuario? usuario)
{

```

```

        if (usuario == null)
        {
            throw new ArgumentNullException("No se encontró un usuario correspondiente a los datos
proporcionados");
        }

        return usuario;
    }

    private static decimal EvaluarMonto(decimal monto)
    {
        if (monto < 0)
        {
            throw new InvalidOperationException("El monto no puede ser negativo");
        }

        return monto;
    }

    // Validación de Oferta
    public void Validar()
    {
    }

    #endregion

    #region Método Equals

    // Sobre escritura del metodo Equals que es usado por Contains
    public override bool Equals(object? obj)
    {
        if (obj != null && obj is Oferta)
        {
            Oferta oferta = (Oferta)obj;

            return Usuario == oferta.Usuario;
        }
    }

```

```

        return false;
    }
    #endregion
}
}

```

PUBLICACION.CS

```

using LogicaNegocio.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace LogicaNegocio
{
    public class Publicacion : IValidate
    {
        #region Atributos de la clase
        private int _id;

        private static int s_ultId = 0; // Inicializado con el id siguiente a la ultima precarga

        private string _nombre = string.Empty; // Inicializado con una cadena vacía

        private string _estado = string.Empty; // Inicializado con una cadena vacía

        private DateTime _fecha = DateTime.Now; // Inicializado con la fecha actual

        private List<Articulo> _articulos = new List<Articulo>(); // Inicializado con una lista vacía

```

```
private Cliente? _cliente; // Inicializado con una instancia por defecto

private Administrador? _administrador; // Inicializado con una instancia por defecto

private DateTime _fechaFin = DateTime.Now; // Inicializado con la fecha actual

#endregion
```

```
#region Propiedades

public int Id

{
    get { return _id; } // Solo lectura, asignado internamente.
}

public string Nombre

{
    get { return _nombre; }

    set { _nombre = EvaluarNombre(value); }
}

public string Estado

{
    get { return _estado; }

    set { _estado = EvaluarEstado(value); }
}

public DateTime Fecha

{
    get { return _fecha; }

    set { _fecha = value; }
}

public DateTime FechaFin

{
    get { return _fechaFin; }

    set { _fechaFin = value; }
```

```

    }

    public List<Articulo> Articulos
    {
        get { return _articulos; }
        set { _articulos = value; }
    }

    public Cliente? Cliente
    {
        get { return _cliente; }
        set { _cliente = value; }
    }

    public Administrador? Administrador
    {
        get { return _administrador; }
        set { _administrador = value; }
    }
}

#endregion

#region Constructor

    public Publicacion(string nombre, string estado, DateTime fecha, List<Articulo> articulos, Cliente?
cliente, Administrador? administrador, DateTime fechaFin)
    {
        _id = Publicacion.s_ultId; // Asigna el ID único

        Publicacion.s_ultId++; // Incrementa el ID único

        Nombre = nombre;

        Estado = estado;

        Fecha = fecha;

        Articulos = articulos;

        Cliente = cliente;

        Administrador = administrador;
    }
}

```



```

        FechaFin = fechaFin;
    }

#endregion

#region Validación
// Evaluaciones
private static string EvaluarNombre(string nombre)
{
    if (string.IsNullOrEmpty(nombre))
    {
        throw new ArgumentException("El nombre no puede ser vacío");
    }

    return nombre;
}

private static string EvaluarEstado(string estado)
{
    if (estado != "ABIERTA" && estado != "CERRADA" && estado != "CANCELADA")
    {
        throw new ArgumentException("El estado de la publicacion tiene que ser ABIERTA, CERRADA o CANCELADA");
    }

    return estado;
}

// Validación de Publicacion, es virtual ya que le hereda a otras clases
public virtual void Validar()
{
    if (FechaFin <= Fecha && FechaFin != DateTime.MinValue)
    {
        throw new InvalidOperationException("La fecha de fin debe ser posterior a la fecha de inicio.");
    }
}

```

```

    }
}

#endregion

#region Método Equals
// Sobre escritura del metodo Equals que es usado por Contains
public override bool Equals(object? obj)
{
    if (obj != null && obj is Publicacion)
    {
        Publicacion publicacion = (Publicacion)obj;
        return Nombre == publicacion.Nombre;
    }
    return false;
}

#endregion
}
}

```

SISTEMA.CS

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

using System.Timers;

using static System.Runtime.InteropServices.JavaScript.JSType;

namespace LogicaNegocio

{

    public class Sistema

    {

        #region Constructor

        // Atributos de la clase con propiedades automaticas (shortHand)

        private List<Usuario> _usuarios { get; set; }

        private List<Publicacion> _publicaciones { get; set; }

        private List<Articulo> _articulos { get; set; }

        // Ejecucion principal

        public Sistema()

        {

            _usuarios = new List<Usuario>();

            _publicaciones = new List<Publicacion>();

            _articulos = new List<Articulo>();

            PrecargaArticulo();

            PrecargaUsuario();

        }

    }

}
```

```

        PrecargarPublicacion();

        PrecargaOferta();
    }

#endregion

/// <summary>
/// El Parseo de datos sirve para modificar los datos ingresados por el usuario
/// o modificar datos para mostrarlos al usuario.
/// La funcion Parseold permite obtener una lista de Ids a partir de una cadena de texto
/// La funcion ParseoArticulo permite obtener una cadena de texto a partir de una lista de articulos
/// </summary>
#region Utilidades
#region Universal

// Convierte en una lista de ids el string pasado por parametros
public List<int> ParseoInt(string intsCrudos)
{
    List<int> lista_ints = new List<int>(); // Crea una lista de los ids ingresados
    string[] ids = intsCrudos.Split(';'); // Crea un array de los ids

    for (int i = 0; i < ids.Length; i++) // Recorre todos los elementos de ids
    {
        lista_ints.Add(int.Parse(ids[i].Trim())); // Remueve los espacios, transforma a int y añade a la
lista el id
    }

    return lista_ints;
}

// Convierte en una lista de nombres el string pasado por parametros
public List<string> ParseoString(string stringsCrudos)
{
    List<string> listaStrings = new List<string>(); // Crea una lista de los nombres ingresados

```

```

string[] nombres = stringsCrudos.Split(','); // Crea un array de los nombres

for (int i = 0; i < nombres.Length; i++) // Recorre todos los elementos de nombres
{
    listaStrings.Add(nombres[i].Trim()); // Remueve los espacios y añade a la lista el nombre
}

return listaStrings;
}

#endregion

#region Artículo

// Convierte en string los ids de la lista de articulos pasada por parametros
public string ParseoArticulo(List<Articulo> articulos)
{
    string ids_articulos = string.Empty;

    for (int i = 0; i < articulos.Count; i++)
    {
        ids_articulos += $"{articulos[i].Id}, ";
    }

    if (articulos.Count > 0)
    {
        // Quitamos la , del final de los ids
        ids_articulos = ids_articulos.Substring(0, ids_articulos.Length - 2);
    }

    return ids_articulos;
}

#endregion

#region Oferta

// Convierte en string los ids de la lista de ofertas pasada por parametros

```

```

public string ParseoOferta(List<Oferta> ofertas)
{
    string ids_ofertas = string.Empty;

    for (int i = 0; i < ofertas.Count; i++)
    {
        ids_ofertas += $"{ofertas[i].Id}, ";
    }

    if (ofertas.Count > 0)
    {
        // Quitamos la , del final de los ids
        ids_ofertas = ids_ofertas.Substring(0, ids_ofertas.Length - 2);
    }

    return ids_ofertas;
}

#endregion

#endregion

/// <summary>
/// Las lista son utilizadas en todas las funciones de impresión.
/// Por ejemplo si queremos imprimir clientes, debemos pasarle a la función
/// imprimirUsuario una lista de clientes.
/// Esta lista se puede conseguir con:
/// ObtenerCliente, almacena en una lista todos los usuarios que sean clientes
/// obtenerClientePorId, almacena en una lista los clientes de ids determinados
/// obtenerClientePorNombre, almacena en una lista los clientes de nombres determinados
/// </summary>

#region Obtención de listas

#region Articulo

```

```

public List<Articulo> ObtenerArticulos()
{
    bool hayArticulo = false;

    List<Articulo> articulos = new List<Articulo>(); // Inicializamos la lista que contendrá los
    artículos

    for (int i = 0; i < _articulos.Count; i++)
    {
        hayArticulo = true;

        articulos.Add(_articulos[i]); // Se añade cualquier artículo a la lista artículos
    }

    if (!hayArticulo)
    {
        throw new ArgumentException("No hay ningún artículo en el sistema");
    }

    return articulos;
}

public List<Articulo> ObtenerArticuloPorId(List<int> ids)
{
    bool hayArticulo = false;

    List<Articulo> articulos = new List<Articulo>(); // Inicializamos la lista que contendrá los
    artículos

    for (int i = 0; i < _articulos.Count; i++)
    {
        if (ids.Contains(_articulos[i].Id)) // Si la lista de ids contiene algún artículo
        {
            hayArticulo = true;

            articulos.Add(_articulos[i]); // Se añade el artículo a la lista artículos
        }
    }

    if (!hayArticulo)

```

```

    {
        throw new ArgumentException("No hay ningún artículo con los ids proporcionados");
    }

    return articulos;
}

public List<Articulo> ObtenerArticuloPorNombre(List<string> nombres)
{
    bool hayArticulo = false;

    List<Articulo> articulos = new List<Articulo>(); // Inicializamos la lista que contendrá los
    artículos

    for (int i = 0; i < _articulos.Count; i++)
    {
        if (nombres.Contains(_articulos[i].Nombre)) // Si la lista de nombres contiene algún artículo
        {
            hayArticulo = true;

            articulos.Add(_articulos[i]); // Se añade el artículo a la lista artículos
        }
    }

    if (!hayArticulo)
    {
        throw new ArgumentException("No hay ningún artículo con los nombres proporcionados");
    }

    return articulos;
}

public List<Articulo> ObtenerArticuloPorCategoria(List<string> categorias)
{
    bool hayArticulo = false;

    List<Articulo> articulos = new List<Articulo>(); // Inicializamos la lista que contendrá los
    artículos

    for (int i = 0; i < _articulos.Count; i++)

```



```

{
    if (categorias.Contains(_articulos[i].Categoria)) // Si la lista de nombres contiene algún artículo
    {
        hayArticulo = true;

        articulos.Add(_articulos[i]); // Se añade el artículo a la lista artículos
    }
}

if (!hayArticulo)
{
    throw new ArgumentException("No hay ningún artículo con las categoría proporcionada");
}

return articulos;
}

#endregion

#region Publicacion

public List<Publicacion> ObtenerPublicaciones(bool esUnicamenteVenta, bool
esUnicamenteSubasta)
{
    bool hayVenta = false;

    bool haySubasta = false;

    List<Publicacion> publicaciones = new List<Publicacion>(); // Inicializamos la lista que
contendrá las publicaciones

    for (int i = 0; i < _publicaciones.Count; i++)
    {
        if ((esUnicamenteVenta && !esUnicamenteSubasta) || (!esUnicamenteVenta &&
!esUnicamenteSubasta))
        {
            if (_publicaciones[i] is Venta venta)
            {
                hayVenta = true;
            }
        }
    }
}

```

```

        publicaciones.Add(venta); // Se añade la venta a la lista publicaciones
    }
}

if ((esUnicamenteSubasta && !esUnicamenteVenta) || (!esUnicamenteVenta &&
!esUnicamenteSubasta))
{
    if (_publicaciones[i] is Subasta subasta)
    {
        haySubasta = true;

        publicaciones.Add(subasta); // Se añade la subasta a la lista publicaciones
    }
}

if (!hayVenta && !haySubasta)
{
    throw new ArgumentException("No hay ninguna publicación en el sistema");
}

if (!hayVenta && esUnicamenteVenta)
{
    throw new ArgumentException("No hay ninguna venta en el sistema");
}

if (!haySubasta && esUnicamenteSubasta)
{
    throw new ArgumentException("No hay ninguna subasta en el sistema");
}

return publicaciones;
}

public List<Publicacion> ObtenerPublicacionPorId(List<int> ids, bool esUnicamenteVenta, bool
esUnicamenteSubasta)
{

```

```

bool hayVenta = false;

bool haySubasta = false;

List<Publicacion> publicaciones = new List<Publicacion>(); // Inicializamos la lista que
contendrá las publicaciones

for (int i = 0; i < _publicaciones.Count; i++)
{
    if (ids.Contains(_publicaciones[i].Id)) // Si la lista de ids contiene alguna publicacion
    {
        if ((esUnicamenteVenta && !esUnicamenteSubasta) || (!esUnicamenteVenta &&
!esUnicamenteSubasta))
        {
            if (_publicaciones[i] is Venta venta)
            {
                hayVenta = true;

                publicaciones.Add(venta); // Se añade la venta a la lista publicaciones
            }
        }

        if ((esUnicamenteSubasta && !esUnicamenteVenta) || (!esUnicamenteVenta &&
!esUnicamenteSubasta))
        {
            if (_publicaciones[i] is Subasta subasta)
            {
                haySubasta = true;

                publicaciones.Add(subasta); // Se añade la subasta a la lista publicaciones
            }
        }
    }
}

if (!hayVenta && !haySubasta)
{

```

```

        throw new ArgumentException("No hay publicaciones con los ids proporcionados");
    }

    if (!hayVenta && esUnicamenteVenta)
    {
        throw new ArgumentException("No hay ventas con los ids proporcionados");
    }

    if (!haySubasta && esUnicamenteSubasta)
    {
        throw new ArgumentException("No hay subastas con los ids proporcionados");
    }

    return publicaciones;
}

public Publicacion? ObtenerPublicacionPorId(int id, bool esUnicamenteVenta, bool
esUnicamenteSubasta)
{
    bool hayVenta = false;

    bool haySubasta = false;

    Publicacion? publicacion = null;

    int indice = 0;

    while (indice < _publicaciones.Count && !hayVenta && !haySubasta)
    {
        if (id == _publicaciones[indice].Id) // Si la lista de ids contiene alguna publicación
        {
            if ((esUnicamenteVenta && !esUnicamenteSubasta) || (!esUnicamenteVenta &&
!esUnicamenteSubasta))
            {
                if (_publicaciones[indice] is Venta venta)
                {
                    hayVenta = true;

                    publicacion = _publicaciones[indice]; // Se asigna la publicación

```

```

        }
    }

    if ((esUnicamenteSubasta && !esUnicamenteVenta) || (!esUnicamenteVenta &&
!esUnicamenteSubasta))
    {
        if (_publicaciones[indice] is Subasta subasta)
        {
            haySubasta = true;

            publicacion = _publicaciones[indice]; // Se asigna la publicación
        }
    }

    indice++;
}

if (!hayVenta && !haySubasta)
{
    throw new ArgumentException("No hay ninguna publicación con el id proporcionado");
}

if (!hayVenta && esUnicamenteVenta)
{
    throw new ArgumentException("No hay ninguna venta con el id proporcionado");
}

if (!haySubasta && esUnicamenteSubasta)
{
    throw new ArgumentException("No hay ninguna subasta con el id proporcionado");
}

return publicacion;
}

public List<Publicacion> ObtenerPublicacionPorNombre(List<string> nombres, bool
esUnicamenteVenta, bool esUnicamenteSubasta)

```

```

{
    bool hayVenta = false;

    bool haySubasta = false;

    List<Publicacion> publicaciones = new List<Publicacion>(); // Inicializamos la lista que
    contendrá las publicaciones

    for (int i = 0; i < _publicaciones.Count; i++)
    {
        if (nombres.Contains(_publicaciones[i].Nombre)) // Si la lista de nombres contiene alguna
        publicación
        {
            if ((esUnicamenteVenta && !esUnicamenteSubasta) || (!esUnicamenteVenta &&
            !esUnicamenteSubasta))
            {
                if (_publicaciones[i] is Venta venta)
                {
                    hayVenta = true;

                    publicaciones.Add(venta); // Se añade la venta a la lista publicaciones
                }
            }

            if ((esUnicamenteSubasta && !esUnicamenteVenta) || (!esUnicamenteVenta &&
            !esUnicamenteSubasta))
            {
                if (_publicaciones[i] is Subasta subasta)
                {
                    haySubasta = true;

                    publicaciones.Add(subasta); // Se añade la subasta a la lista publicaciones
                }
            }
        }
    }

    if (!hayVenta && !haySubasta)

```

```

    {
        throw new ArgumentException("No hay publicaciones con los nombres proporcionados");
    }

    if (!hayVenta && esUnicamenteVenta)
    {
        throw new ArgumentException("No hay ventas con los nombres proporcionados");
    }

    if (!haySubasta && esUnicamenteSubasta)
    {
        throw new ArgumentException("No hay subastas con los nombres proporcionados");
    }

    return publicaciones;
}

public Publicacion? ObtenerPublicacionPorNombre(string nombre, bool esUnicamenteVenta, bool
esUnicamenteSubasta)
{
    bool hayVenta = false;
    bool haySubasta = false;
    Publicacion? publicacion = null;
    int indice = 0;
    while (indice < _publicaciones.Count && !hayVenta && !haySubasta)
    {
        if (nombre.Contains(_publicaciones[indice].Nombre)) // Si la lista de nombres contiene alguna
publicación
        {
            if ((esUnicamenteVenta && !esUnicamenteSubasta) || (!esUnicamenteVenta &&
!esUnicamenteSubasta))
            {
                if (_publicaciones[indice] is Venta venta)
                {

```

```

        hayVenta = true;

        publicacion = _publicaciones[indice]; // Se asigna la publicación
    }
}

if ((esUnicamenteSubasta && !esUnicamenteVenta) || (!esUnicamenteVenta &&
!esUnicamenteSubasta))
{
    if (_publicaciones[indice] is Subasta subasta)
    {
        haySubasta = true;

        publicacion = _publicaciones[indice]; // Se asigna la publicación
    }
}

indice++;
}

if (!hayVenta && !haySubasta)
{
    throw new ArgumentException("No hay ninguna publicación con el nombre proporcionado");
}

if (!hayVenta && esUnicamenteVenta)
{
    throw new ArgumentException("No hay ninguna venta con el nombre proporcionado");
}

if (!haySubasta && esUnicamenteSubasta)
{
    throw new ArgumentException("No hay ninguna subasta con el nombre proporcionado");
}

return publicacion;
}

```



```

    public List<Publicacion> ObtenerPublicacionPorFecha(DateTime fechaInicio, DateTime
fechaFin, bool esUnicamenteVenta, bool esUnicamenteSubasta)

    {

        bool hayVenta = false;

        bool haySubasta = false;

        List<Publicacion> publicaciones = new List<Publicacion>(); // Inicializamos la lista que
contendrá las publicaciones

        for (int i = 0; i < _publicaciones.Count; i++)

        {

            if (_publicaciones[i].Fecha > fechaInicio && _publicaciones[i].Fecha < fechaFin) // Si la lista de
nombres contiene alguna publicación

            {

                if ((esUnicamenteVenta && !esUnicamenteSubasta) || (!esUnicamenteVenta &&
!esUnicamenteSubasta))

                {

                    if (_publicaciones[i] is Venta venta)

                    {

                        hayVenta = true;

                        publicaciones.Add(_publicaciones[i]); // Se asigna la publicación

                    }

                }

                if ((esUnicamenteSubasta && !esUnicamenteVenta) || (!esUnicamenteVenta &&
!esUnicamenteSubasta))

                {

                    if (_publicaciones[i] is Subasta subasta)

                    {

                        haySubasta = true;

                        publicaciones.Add(_publicaciones[i]); // Se asigna la publicación

                    }

                }

            }

        }

    }

```

```

    }

    if (!hayVenta && !haySubasta)
    {
        throw new ArgumentException("No hay publicaciones entre las fechas proporcionadas");
    }

    if (!hayVenta && esUnicamenteVenta)
    {
        throw new ArgumentException("No hay ventas entre las fechas proporcionadas");
    }

    if (!haySubasta && esUnicamenteSubasta)
    {
        throw new ArgumentException("No hay subastas entre las fechas proporcionadas");
    }

    return publicaciones;
}

#endregion

#region Usuario

public List<Usuario> ObtenerUsuarios(bool esUnicamenteCliente, bool
esUnicamenteAdministrador)
{
    bool hayCliente = false;

    bool hayAdministrador = false;

    List<Usuario> usuarios = new List<Usuario>(); // Inicializamos la lista que contendrá los usuarios

    for (int i = 0; i < _usuarios.Count; i++)
    {
        if ((esUnicamenteCliente && !esUnicamenteAdministrador) || (!esUnicamenteCliente &&
!esUnicamenteAdministrador))
        {
            if (_usuarios[i] is Cliente cliente)
            {

```

```

        hayCliente = true;

        usuarios.Add(cliente); // Se añade el cliente a la lista usuarios
    }
}

if ((esUnicamenteAdministrador && !esUnicamenteCliente) || (!esUnicamenteCliente &&
!esUnicamenteAdministrador))
{
    if (_usuarios[i] is Administrador administrador)
    {
        hayAdministrador = true;

        usuarios.Add(administrador); // Se añade el administrador a la lista usuarios
    }
}

if (!hayCliente && !hayAdministrador)
{
    throw new ArgumentException("No hay ningún usuario en el sistema");
}

if (!hayCliente && esUnicamenteCliente)
{
    throw new ArgumentException("No hay ningún cliente en el sistema");
}

if (!hayAdministrador && esUnicamenteAdministrador)
{
    throw new ArgumentException("No hay ningún administrador en el sistema");
}

return usuarios;
}

public List<Usuario> ObtenerUsuarioPorId(List<int> ids, bool esUnicamenteCliente, bool
esUnicamenteAdministrador)

```

```

{
    bool hayCliente = false;

    bool hayAdministrador = false;

    List<Usuario> usuarios = new List<Usuario>(); // Inicializamos la lista que contendrá los usuarios

    for (int i = 0; i < _usuarios.Count; i++)
    {
        if (ids.Contains(_usuarios[i].Id)) // Si la lista de ids contiene algún usuario
        {
            if ((esUnicamenteCliente && !esUnicamenteAdministrador) || (!esUnicamenteCliente &&
!esUnicamenteAdministrador))
            {
                if (_usuarios[i] is Cliente cliente)
                {
                    hayCliente = true;

                    usuarios.Add(cliente); // Se añade el cliente a la lista usuarios
                }
            }

            if ((esUnicamenteAdministrador && !esUnicamenteCliente) || (!esUnicamenteCliente &&
!esUnicamenteAdministrador))
            {
                if (_usuarios[i] is Administrador administrador)
                {
                    hayAdministrador = true;

                    usuarios.Add(administrador); // Se añade el administrador a la lista usuarios
                }
            }
        }
    }

    if (!hayCliente && !hayAdministrador)
    {

```

```

        throw new ArgumentException("No hay usuarios con los ids proporcionados");
    }

    if (!hayCliente && esUnicamenteCliente)
    {
        throw new ArgumentException("No hay clientes con los ids proporcionados");
    }

    if (!hayAdministrador && esUnicamenteAdministrador)
    {
        throw new ArgumentException("No hay administradores con los ids proporcionados");
    }

    return usuarios;
}

public Usuario? ObtenerUsuarioPorId(int id, bool esUnicamenteCliente, bool
esUnicamenteAdministrador)
{
    bool hayCliente = false;

    bool hayAdministrador = false;

    Usuario? usuario = null;

    int indice = 0;

    while (indice < _usuarios.Count && !hayCliente && !hayAdministrador)
    {
        if (id == _usuarios[indice].Id) // Si la lista de ids contiene alguna usuario
        {
            if ((esUnicamenteCliente && !esUnicamenteAdministrador) || (!esUnicamenteCliente &&
!esUnicamenteAdministrador))
            {
                if (_usuarios[indice] is Cliente cliente)
                {
                    hayCliente = true;

                    usuario = cliente; // Se asigna el usuario
                }
            }
        }
    }
}

```

```

        }
    }

    if ((esUnicamenteAdministrador && !esUnicamenteCliente) || (!esUnicamenteCliente &&
!esUnicamenteAdministrador))
    {
        if (_usuarios[indice] is Administrador administrador)
        {
            hayAdministrador = true;
            usuario = administrador; // Se asigna el usuario
        }
    }

    indice++;
}

if (!hayCliente && !hayAdministrador)
{
    throw new ArgumentException("No hay ningún usuario con el id proporcionado");
}

if (!hayCliente && esUnicamenteCliente)
{
    throw new ArgumentException("No hay ningún cliente con el id proporcionado");
}

if (!hayAdministrador && esUnicamenteAdministrador)
{
    throw new ArgumentException("No hay ningún administrador con el id proporcionado");
}

return usuario;
}

public List<Usuario> ObtenerUsuarioPorNombre(List<string> nombres, bool
esUnicamenteCliente, bool esUnicamenteAdministrador)

```

```

{
    bool hayCliente = false;

    bool hayAdministrador = false;

    List<Usuario> usuarios = new List<Usuario>(); // Inicializamos la lista que contendrá los usuarios

    for (int i = 0; i < _usuarios.Count; i++)
    {
        if (nombres.Contains(_usuarios[i].Nombre)) // Si la lista de nombres contiene algún usuario
        {
            if ((esUnicamenteCliente && !esUnicamenteAdministrador) || (!esUnicamenteCliente &&
!esUnicamenteAdministrador))
            {
                if (_usuarios[i] is Cliente cliente)
                {
                    hayCliente = true;

                    usuarios.Add(cliente); // Se añade el cliente a la lista usuarios
                }
            }

            if ((esUnicamenteAdministrador && !esUnicamenteCliente) || (!esUnicamenteCliente &&
!esUnicamenteAdministrador))
            {
                if (_usuarios[i] is Administrador administrador)
                {
                    hayAdministrador = true;

                    usuarios.Add(administrador); // Se añade el administrador a la lista usuarios
                }
            }
        }
    }

    if (!hayCliente && !hayAdministrador)
    {

```

```

        throw new ArgumentException("No hay usuarios con los nombres proporcionados");
    }

    if (!hayCliente && esUnicamenteCliente)
    {
        throw new ArgumentException("No hay clientes con los nombres proporcionados");
    }

    if (!hayAdministrador && esUnicamenteAdministrador)
    {
        throw new ArgumentException("No hay administradores con los nombres proporcionados");
    }

    return usuarios;
}

public Usuario? ObtenerUsuarioPorNombre(string nombre, bool esUnicamenteCliente, bool
esUnicamenteAdministrador)
{
    bool hayCliente = false;

    bool hayAdministrador = false;

    Usuario? usuario = null;

    int indice = 0;

    while (indice < _usuarios.Count && !hayCliente && !hayAdministrador)
    {
        if (nombre.Contains(_usuarios[indice].Nombre)) // Si la lista de nombres contiene algún
usuario
        {
            if ((esUnicamenteCliente && !esUnicamenteAdministrador) || (!esUnicamenteCliente &&
!esUnicamenteAdministrador))
            {
                if (_usuarios[indice] is Cliente cliente)
                {
                    hayCliente = true;

```



```

        usuario = _usuarios[indice]; // Se asigna el usuario
    }
}

if ((esUnicamenteAdministrador && !esUnicamenteCliente) || (!esUnicamenteCliente &&
!esUnicamenteAdministrador))
{
    if (_usuarios[indice] is Administrador administrador)
    {
        hayAdministrador = true;

        usuario = _usuarios[indice]; // Se asigna el usuario
    }
}

indice++;
}

if (!hayCliente && !hayAdministrador)
{
    throw new ArgumentException("No hay ningún usuario con el nombre proporcionado");
}

if (!hayCliente && esUnicamenteCliente)
{
    throw new ArgumentException("No hay ningún cliente con el nombre proporcionado");
}

if (!hayAdministrador && esUnicamenteAdministrador)
{
    throw new ArgumentException("No hay ningún administrador con el nombre proporcionado");
}

return usuario;
}

#endregion

```

```

#region Oferta

public List<Oferta> ObtenerOfertas(Publicacion? publicacion)
{
    bool hayOferta = false;

    List<Oferta> ofertas = new List<Oferta>(); // Inicializamos la lista que contendrá las ofertas

    if (publicacion is Subasta subasta)
    {
        hayOferta = true;

        ofertas = subasta.Ofertas; // Se añade cualquier oferta a la lista ofertas
    }

    if (!hayOferta)
    {
        throw new ArgumentException("La subasta no contiene ofertas");
    }

    return ofertas;
}

#endregion

#endregion

```

```

/// <summary>

/// Las funciones de alta se encargan de llamar a los constructores de las

/// diferentes clases y pasar los parametros obtenidos en Program.

/// </summary>

#region Altas

#region Artículo

public void AltaArticulo(string nombre, decimal precio, string categoria)
{
    Articulo nuevoArticulo = new Articulo(nombre, precio, categoria);

    // Validación de la relacion entre los datos ingresados

```

```

nuevoArticulo.Validar();

// Si los datos son validos entonces se registra el Articulo
if (!_articulos.Contains(nuevoArticulo))
{
    _articulos.Add(nuevoArticulo);
}
else
{
    throw new ArgumentException("Ya existe un articulo registrado con el nombre proporcionado");
}
}

#endregion

#region Publicacion

public void AltaPublicacion(string nombre, string estado, DateTime fecha, List<Articulo> articulos,
Cliente? cliente, Administrador? administrador, DateTime fechaFin)
{
    Publicacion nuevaPublicacion = new Publicacion(nombre, estado, fecha, articulos, cliente,
administrador, fechaFin);

    // Validación de la relacion entre los datos ingresados
    nuevaPublicacion.Validar();

    // Si los datos son validos entonces se registra la Publicación
    if (!_publicaciones.Contains(nuevaPublicacion))
    {
        _publicaciones.Add(nuevaPublicacion);
    }
    else
    {
        throw new ArgumentException("Ya existe una publicacion registrada con el nombre
proporcionado");
    }
}

```

```

    }

    public void AltaVenta(string nombre, string estado, DateTime fecha, List<Articulo> articulos,
    Cliente? cliente, Administrador? administrador, DateTime fechaFin, bool ofertaRelampago)

    {

        Venta nuevaVenta = new Venta(nombre, estado, fecha, articulos, cliente, administrador, fechaFin,
ofertaRelampago);

        // Validación de la relacion entre los datos ingresados

        nuevaVenta.Validar();

        // Si los datos son validos entonces se registra la Venta

        if (!_publicaciones.Contains(nuevaVenta))

        {

            _publicaciones.Add(nuevaVenta);

        }

        else

        {

            throw new ArgumentException("Ya existe una publicacion registrada con el nombre
proporcionado");

        }

    }

    public void AltaSubasta(string nombre, string estado, DateTime fecha, List<Articulo> articulos,
    Cliente? cliente, Administrador? administrador, DateTime fechaFin, List<Oferta> ofertas)

    {

        Subasta nuevaSubasta = new Subasta(nombre, estado, fecha, articulos, cliente, administrador,
fechaFin, ofertas);

        // Validación de la relacion entre los datos ingresados

        nuevaSubasta.Validar();

        // Si los datos son validos entonces se registra la Subasta

        if (!_publicaciones.Contains(nuevaSubasta))

        {

            _publicaciones.Add(nuevaSubasta);

        }

    }

```

```

        else

        {

            throw new ArgumentException("Ya existe una publicacion registrada con el nombre
proporcionado");

        }

    }

#endregion

#region Usuario

public void AltaUsuario(string nombre, string apellido, string email, string contrasenia)
{

    Usuario nuevoUsuario = new Usuario(nombre, apellido, email, contrasenia);

    // Validación de la relacion entre los datos ingresados

    nuevoUsuario.Validar();

    // Si los datos son validos entonces se registra el Usuario

    if (!_usuarios.Contains(nuevoUsuario))

    {

        _usuarios.Add(nuevoUsuario);

    }

    else

    {

        throw new ArgumentException("Ya existe un usuario registrado con el nombre y apellido
proporcionados");

    }

}

public void AltaCliente(string nombre, string apellido, string email, string contrasenia, decimal
saldo)

{

    Cliente nuevoCliente = new Cliente(nombre, apellido, email, contrasenia, saldo);

    // Validación de la relacion entre los datos ingresados

    nuevoCliente.Validar();

```

```

// Si los datos son validos entonces se registra el Cliente
if (!_usuarios.Contains(nuevoCliente))
{
    _usuarios.Add(nuevoCliente);
}
else
{
    throw new ArgumentException("Ya existe un usuario registrado con el nombre y apellido
proporcionados");
}
}

public void AltaAdministrador(string nombre, string apellido, string email, string contrasenia)
{
    Administrador nuevoAdministrador = new Administrador(nombre, apellido, email, contrasenia);
    // Validación de la relacion entre los datos ingresados
    nuevoAdministrador.Validar();
    // Si los datos son validos entonces se registra el Administrador
    if (!_usuarios.Contains(nuevoAdministrador))
    {
        _usuarios.Add(nuevoAdministrador);
    }
    else
    {
        throw new ArgumentException("Ya existe un usuario registrado con el nombre y apellido
proporcionados");
    }
}

#endregion

#region Ofertas

```

```

    public void AltaOferta(Usuario? usuario, Publicacion? publicacion, decimal monto, DateTime
fecha)
    {
        if (publicacion != null && publicacion is Subasta subasta)
        {
            subasta.AltaOferta(usuario, monto, fecha);
        }
        else if (publicacion == null)
        {
            throw new ArgumentNullException("No fue posible acceder a la subasta proporcionada");
        }
        else
        {
            throw new ArgumentException("No fue posible realizar la oferta debido a que no fue posible
identificar la subasta");
        }
    }
#endregion
#endregion

```

```

/// <summary>
/// Las funciones de consulta tienen el objetivo de obtener datos calculados.
/// Por ejemplo ConsultarPrecioVentaDeListaVenta obtiene los precios de las ventas buscadas.
/// Este es un dato calculado ya que es necesario acceder a la venta y sumar el precio de todos sus
articulos.
/// </summary>
#region Consultas
public decimal ConsultarPrecioVenta(Publicacion? publicacion, List<Articulo> articulos)
{
    decimal precio = 0;

```

```

for (int i = 0; i < articulos.Count; i++)
{
    precio += articulos[i].Precio;
}

if (publicacion is Venta venta)
{
    if (venta.OfertaRelampago)
    {
        // se aplica descuento si corresponde a la venta en especifico
        precio = precio * 80 / 100;
    }
}

return precio;
}

public bool HayCliente()
{
    for (int i = 0; i < _usuarios.Count; i++)
    {
        if (_usuarios[i] is Cliente cliente)
        {
            return true;
        }
    }

    return false;
}

#endregion

```

/// <summary>

/// Las precargas son realizadas a travez de las funciones de alta,


```
/// esto se hace de este modo para que el id autoincremental se asigne correctamente
```

```
/// </summary>
```

```
#region Precargas
```

```
#region Artículo
```

```
private void PrecargaArticulo()
```

```
{
```

```
    AltaArticulo("Pelota de football", 450, "Football");
```

```
    AltaArticulo("Camiseta deportiva", 1200, "Deporte");
```

```
    AltaArticulo("Zapatillas treking", 3500, "Treking");
```

```
    AltaArticulo("Raqueta de tenis", 4200, "Tenis");
```

```
    AltaArticulo("Balón de basquetball", 800, "Basquetball");
```

```
    AltaArticulo("Guantes de boxeo", 2200, "Boxeo");
```

```
    AltaArticulo("Casco de ciclismo", 1800, "Ciclismo");
```

```
    AltaArticulo("Saco de dormir", 2300, "Camping");
```

```
    AltaArticulo("Bolsa de gimnasio", 950, "Boxeo");
```

```
    AltaArticulo("Bicicleta de montaña", 15000, "Ciclismo");
```

```
    AltaArticulo("Mochila de trekking", 2100, "Treking");
```

```
    AltaArticulo("Protector solar", 320, "Playa");
```

```
    AltaArticulo("Botella térmica", 750, "Camping");
```

```
    AltaArticulo("Palo de hockey", 1700, "Hokey");
```

```
    AltaArticulo("Pesas ajustables", 3000, "Gimnasio");
```

```
    AltaArticulo("Cinta para correr", 25000, "Gimnasio");
```

```
    AltaArticulo("Guantes de arquero", 900, "Arquería");
```

```
    AltaArticulo("Tabla de surf", 12000, "Surf");
```

```
    AltaArticulo("Canilleras", 600, "Football");
```

```
    AltaArticulo("Traje de neopreno", 5400, "Surf");
```

```
    AltaArticulo("Gafas de natación", 650, "Natación");
```

```
    AltaArticulo("Bola de bowling", 3500, "Bowling");
```

```
    AltaArticulo("Skateboard", 2400, "Skating");
```

AltaArticulo("Patines en línea", 2900, "Patinaaje");

AltaArticulo("Salvavidas", 1200, "Playa");

AltaArticulo("Set de pesas", 4200, "Gimnasio");

AltaArticulo("Cuerda para saltar", 300, "Gimnasio");

AltaArticulo("Bicicleta de carrera", 18500, "Ciclismo");

AltaArticulo("Tobilleras con peso", 850, "Gimnasio");

AltaArticulo("Set de dardos", 400, "Juegos");

AltaArticulo("Bate de baseball", 1900, "Baseball");

AltaArticulo("Bola de volleyball", 850, "Volleyball");

AltaArticulo("Aro de basketball", 2700, "Basketball");

AltaArticulo("Zapatilla de ciclismo", 1900, "Ciclismo");

AltaArticulo("Silla de camping", 1100, "Camping");

AltaArticulo("Sombrilla", 1600, "Playa");

AltaArticulo("Tienda de campaña", 8700, "Camping");

AltaArticulo("Colchoneta de yoga", 1200, "Deporte");

AltaArticulo("Barra de dominadas", 1900, "Gimnasio");

AltaArticulo("Malla", 600, "Ciclismo");

AltaArticulo("Reloj deportivo", 6500, "Deporte");

AltaArticulo("Monopatín eléctrico", 18000, "Ciclismo");

AltaArticulo("Kit de pesca", 3200, "Pesca");

AltaArticulo("Bolsa de golf", 7600, "Golf");

AltaArticulo("Raqueta de badminton", 1600, "Badminton");

AltaArticulo("Patineta longboard", 3300, "Skating");

AltaArticulo("Bola de rugby", 1050, "Rugby");

AltaArticulo("Kit de snorkel", 1800, "Natacion");

AltaArticulo("Camiseta de compresión", 1300, "Deporte");

AltaArticulo("Gorra deportiva", 400, "Deporte");

AltaArticulo("Balón medicinal", 2000, "Salud");

AltaArticulo("Kit de arquería", 9800, "Arquería");

```

        AltaArticulo("Soga de escalada", 5600, "Escalada");

        AltaArticulo("Casco de ski", 3700, "Ski");

        AltaArticulo("Balde", 1050, "Playa");

        AltaArticulo("Gafas de ciclismo", 900, "Ciclismo");

    }

    #endregion

    #region Publicacion

    private void PrecargarPublicacion()

    {

        AltaVenta("Verano en la playa", "ABIERTA", DateTime.ParseExact("05/01/2024", "dd/MM/yyyy",
null), ObtenerArticuloPorId(new List<int> { 11, 24, 35, 54 })), null, null, DateTime.MinValue, false);

        AltaSubasta("Vuelta ciclista", "ABIERTA", DateTime.ParseExact("06/01/2024", "dd/MM/yyyy", null),
ObtenerArticuloPorId(new List<int> { 27, 33, 39 })), null, null, DateTime.MinValue, new List<Oferta>());

        AltaSubasta("Set camping", "ABIERTA", DateTime.ParseExact("21/07/2024", "dd/MM/yyyy", null),
ObtenerArticuloPorId(new List<int> { 7, 34, 36 })), null, null, DateTime.MinValue, new List<Oferta>());

        AltaVenta("Juego gimnasio", "ABIERTA", DateTime.ParseExact("13/12/2024", "dd/MM/yyyy", null),
ObtenerArticuloPorId(new List<int> { 14, 15, 25, 26, 28, 38 })), null, null, DateTime.MinValue, false);

        AltaVenta("Caminata en el bosque", "ABIERTA", DateTime.ParseExact("12/02/2024", "dd/MM/yyyy",
null), ObtenerArticuloPorId(new List<int> { 1, 3, 4, 5 })), null, null, DateTime.MinValue, false);

        AltaVenta("Paseo en bicicleta", "ABIERTA", DateTime.ParseExact("15/03/2024", "dd/MM/yyyy",
null), ObtenerArticuloPorId(new List<int> { 6, 8, 9, 10 })), null, null, DateTime.MinValue, false);

        AltaVenta("Clase de yoga", "ABIERTA", DateTime.ParseExact("22/04/2024", "dd/MM/yyyy", null),
ObtenerArticuloPorId(new List<int> { 12, 13, 16, 18, 20 })), null, null, DateTime.MinValue, false);

        AltaVenta("Día de spa", "ABIERTA", DateTime.ParseExact("30/05/2024", "dd/MM/yyyy", null),
ObtenerArticuloPorId(new List<int> { 21, 22, 23, 29 })), null, null, DateTime.MinValue, false);

        AltaVenta("Concierto al aire libre", "ABIERTA", DateTime.ParseExact("01/08/2024", "dd/MM/yyyy",
null), ObtenerArticuloPorId(new List<int> { 30, 31, 32, 34, 37 })), null, null, DateTime.MinValue, false);

        AltaVenta("Cata de vinos", "ABIERTA", DateTime.ParseExact("10/09/2024", "dd/MM/yyyy", null),
ObtenerArticuloPorId(new List<int> { 40, 41, 42 })), null, null, DateTime.MinValue, false);

        AltaVenta("Taller de pintura", "ABIERTA", DateTime.ParseExact("15/10/2024", "dd/MM/yyyy", null),
ObtenerArticuloPorId(new List<int> { 43, 44, 45, 46 })), null, null, DateTime.MinValue, false);

        AltaVenta("Excursión a la montaña", "ABIERTA", DateTime.ParseExact("25/11/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 47, 48, 49 })), null, null, DateTime.MinValue,
false);
    }

```

```
AltaSubasta("Torneo de ajedrez", "ABIERTA", DateTime.ParseExact("12/03/2024", "dd/MM/yyyy",
null), ObtenerArticuloPorId(new List<int> { 50, 51, 52 }), null, null, DateTime.MinValue, new
List<Oferta>());
```

```
AltaSubasta("Subasta de arte", "ABIERTA", DateTime.ParseExact("20/04/2024", "dd/MM/yyyy",
null), ObtenerArticuloPorId(new List<int> { 51, 53, 54 }), null, null, DateTime.MinValue, new
List<Oferta>());
```

```
AltaSubasta("Rally de coches", "ABIERTA", DateTime.ParseExact("01/06/2024", "dd/MM/yyyy",
null), ObtenerArticuloPorId(new List<int> { 36, 37, 38 }), null, null, DateTime.MinValue, new
List<Oferta>());
```

```
AltaSubasta("Subasta de antigüedades", "ABIERTA", DateTime.ParseExact("15/07/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 29, 20, 21 }), null, null, DateTime.MinValue,
new List<Oferta>());
```

```
AltaSubasta("Concurso de cocina", "ABIERTA", DateTime.ParseExact("05/08/2024", "dd/MM/yyyy",
null), ObtenerArticuloPorId(new List<int> { 42, 43, 44 }), null, null, DateTime.MinValue, new
List<Oferta>());
```

```
AltaSubasta("Maratón de lectura", "ABIERTA", DateTime.ParseExact("12/09/2024", "dd/MM/yyyy",
null), ObtenerArticuloPorId(new List<int> { 45, 46, 47 }), null, null, DateTime.MinValue, new
List<Oferta>());
```

```
AltaSubasta("Competencia de fotografía", "ABIERTA", DateTime.ParseExact("30/10/2024",
"dd/MM/yyyy", null), ObtenerArticuloPorId(new List<int> { 18, 19, 20 }), null, null, DateTime.MinValue,
new List<Oferta>());
```

```
AltaSubasta("Fiesta de disfraces", "ABIERTA", DateTime.ParseExact("15/11/2024", "dd/MM/yyyy",
null), ObtenerArticuloPorId(new List<int> { 21, 22, 23 }), null, null, DateTime.MinValue, new
List<Oferta>());
```

```
}
```

```
#endregion
```

```
#region Usuario
```

```
private void PrecargaUsuario()
```

```
{
```

```
AltaAdministrador("Valentin", "Latorre", "ValentinLatorre@Gmail.com", "Valentin1234");
```

```
AltaAdministrador("Agustin", "Butrico", "AgustinButrico@gmail.com", "Agustin1234");
```

```
AltaCliente("Juan", "Peres", "Juanperes@hmail.com", "Juan1234", 2000);
```

```
AltaCliente("Esteban", "Lopez", "EstebanLopez@hmail.com", "556643", 2000);
```

```
AltaCliente("Carlos", "Medina", "CarlosMedina@hmail.com", "Medina1234", 4500);
```

```
AltaCliente("Mariano", "Morales", "MarianoMorales@hmail.com", "Mariano2", 5000);
```

```

        AltaCliente("Estela", "Rosales", "EstelaRosales@hmail.com", "Rosalia46", 300);

        AltaCliente("Marcos", "Sauce", "MarcosSauce@hmail.com", "Sauce31", 30000);

        AltaCliente("Lucia", "Gomez", "LuciaGomezs@hmail.com", "Lucia1990", 7200);

        AltaCliente("Rodrigo", "Barrios", "RodrigoBarrios@hmail.com", "RodrigoBarrios12", 900);

        AltaCliente("Pepe", "Argento", "PepeArgento@gmail.com", "PepeArgento1113", 3300);

        AltaCliente("Felipe", "Castañeda", "FelipeCastañeda@gmail.com", "FeliCastañeda032", 3300);

    }

    #endregion

    #region Oferta

    private void PrecargaOferta()

    {

        AltaOferta(ObtenerUsuarioPorId(3, true, false), ObtenerPublicacionPorId(2, false, true), 100,
DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

        AltaOferta(ObtenerUsuarioPorId(6, true, false), ObtenerPublicacionPorId(1, false, true), 1500,
DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

        AltaOferta(ObtenerUsuarioPorId(4, true, false), ObtenerPublicacionPorId(1, false, true), 3400,
DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

        AltaOferta(ObtenerUsuarioPorId(8, true, false), ObtenerPublicacionPorId(1, false, true), 3500,
DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

        AltaOferta(ObtenerUsuarioPorId(3, true, false), ObtenerPublicacionPorId(1, false, true), 20000,
DateTime.ParseExact("24/07/2024", "dd/MM/yyyy", null));

    }

    #endregion

    #endregion

}
}

```

SUBASTA.CS

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LogicaNegocio
{
    public class Subasta : Publicacion
    {
        #region Atributos de la clase

        private List<Oferta> _ofertas = new List<Oferta>(); // Inicializado con una lista vacía

        #endregion

        #region Propiedades

        public List<Oferta> Ofertas
        {
            get { return _ofertas; }

            set { _ofertas = value; }

        }

        #endregion

        #region Constructor

        public Subasta(string nombre, string estado, DateTime fecha, List<Articulo> articulos, Cliente? cliente, Administrador? administrador, DateTime fechaFin, List<Oferta> ofertas)
            : base(nombre, estado, fecha, articulos, cliente, administrador, fechaFin) // Llamada al constructor de la clase base (Publicacion)
        {
            Ofertas = ofertas;
        }
    }
}
```

```

    }

#endregion

#region Validación
// Validación de Subasta, hereda de Publicacion
public override void Validar()
{
}
#endregion

#region Método Equals
// Sobre escritura del metodo Equals que es usado por Contains
public override bool Equals(object? obj)
{
    if (obj != null && obj is Subasta)
    {
        Subasta subasta = (Subasta)obj;

        return Nombre == subasta.Nombre;
    }

    return false;
}
#endregion

#region Alta
public void AltaOferta(Usuario? usuario, decimal monto, DateTime fecha)
{
    Oferta oferta = new Oferta(usuario, monto, fecha); // Crea una oferta con el constructor de Oferta

    if (usuario != null && !Ofertas.Contains(oferta)) // Utilizando el Equals de Oferta valida que un
        usuario no haga más de una oferta
    {

```

```
        Ofertas.Add(oferta); // Añade a la lista _ofertas
    }
}
#endregion
}
}
```


USUARIO.CS

```
using LogicaNegocio.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LogicaNegocio
{
    public class Usuario : IValidate
    {
        #region Atributos de la clase

        private int _id;

        private static int s_ultId = 0; // Inicializado con el id siguiente a la ultima precarga

        private string _nombre = string.Empty; // Inicializado con una cadena vacía

        private string _apellido = string.Empty; // Inicializado con una cadena vacía

        private string _email = string.Empty; // Inicializado con una cadena vacía

        private string _contrasenia = string.Empty; // Inicializado con una cadena vacía

        #endregion

        #region Propiedades

        public int Id
        {
            get { return _id; } // Solo lectura, asignado internamente.
        }

        public string Nombre
        {
            get { return _nombre; }
```

```

        set { _nombre = EvaluarNombre(value); }
    }

    public string Apellido
    {
        get { return _apellido; }

        set { _apellido = EvaluarApellido(value); }
    }

    public string Email
    {
        get { return _email; }

        set { _email = EvaluarEmail(value); }
    }

    public string Contraseña
    {
        get { return _contraseña; }

        set { _contraseña = EvaluarContraseña(value); }
    }
}

#endregion

#region Constructor

public Usuario(string nombre, string apellido, string email, string contraseña)
{
    _id = Usuario.s_ultId; // Asigna el ID único

    Usuario.s_ultId++; // Incrementa el ID único

    Nombre = nombre;

    Apellido = apellido;

    Email = email;

    Contraseña = contraseña;
}

#endregion

```

#region Validación

// Evaluaciones

private static string EvaluarNombre(string nombre)

```
{  
    if (string.IsNullOrEmpty(nombre))  
    {  
        throw new ArgumentException("El nombre no puede ser vacío");  
    }  
    return nombre;  
}
```

private static string EvaluarApellido(string apellido)

```
{  
    if (string.IsNullOrEmpty(apellido))  
    {  
        throw new ArgumentException("El apellido no puede ser vacío");  
    }  
    return apellido;  
}
```

private static string EvaluarEmail(string email)

```
{  
    if (string.IsNullOrEmpty(email))  
    {  
        throw new ArgumentException("El email no puede ser vacío");  
    }  
    if (email.IndexOf('@') == -1)  
    {  
        throw new ArgumentException("El email debe pertenecer a un domino (debe tener @)");  
    }  
}
```

```

        return email;
    }

    private static string EvaluarContrasenia(string contrasenia)
    {
        if (string.IsNullOrEmpty(contrasenia))
        {
            throw new ArgumentException("El contrasenia no puede ser vacío");
        }

        return contrasenia;
    }

    // Validación de Usuario, es virtual ya que le hereda a otras clases
    public virtual void Validar()
    {
    }

    #endregion

    #region Método Equals

    // Sobre escritura del metodo Equals que es usado por Contains
    public override bool Equals(object? obj)
    {
        if (obj != null && obj is Usuario)
        {
            Usuario usuario = (Usuario)obj;

            return Nombre == usuario.Nombre && Apellido == usuario.Apellido;
        }

        return false;
    }

    #endregion
}

```

VENTA.CS

```
using LogicaNegocio.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LogicaNegocio
{
    public class Venta : Publicacion
    {
        #region Atributos de la clase

        private bool _ofertaRelampago = false; // Inicializado en falso

        #endregion

        #region Propiedades

        public bool OfertaRelampago
        {
            get { return _ofertaRelampago; }

            set { _ofertaRelampago = value; }
        }

        #endregion

        #region Constructor

        public Venta(string nombre, string estado, DateTime fecha, List<Articulo> articulos, Cliente? cliente, Administrador? administrador, DateTime fechaFin, bool ofertaRelampago)
```

: base(nombre, estado, fecha, articulos, cliente, administrador, fechaFin) // Llamada al constructor de la clase base (Publicacion)

```
{
    OfertaRelampago = ofertaRelampago;
}

#endregion

#region Validación
// Validación de Venta, hereda de Publicacion
public override void Validar()
{
}

#endregion

#region Método Equals
// Sobre escritura del metodo Equals que es usado por Contains
public override bool Equals(object? obj)
{
    if (obj != null && obj is Venta)
    {
        Venta venta = (Venta)obj;
        return Nombre == venta.Nombre;
    }
    return false;
}

#endregion
}
}
```