Universidad ORT Uruguay
Facultad de Ingeniería
Escuela de Tecnología

DOCUMENTACIÓN OBLIGATORIO 1 PROGRAMACIÓN 2

Álvaro Pérez– 234980

Jonatan Miranda– 294433

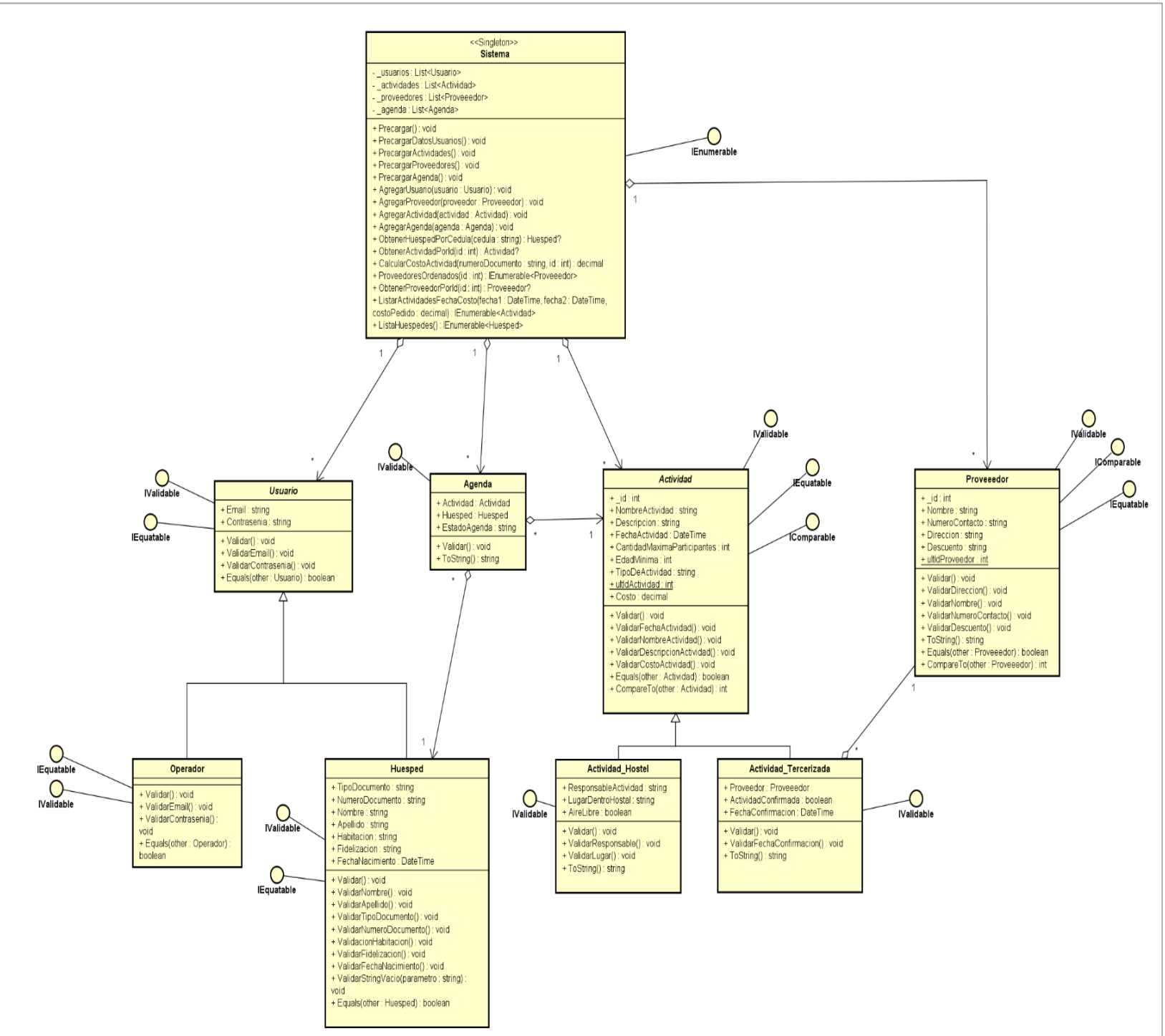
Grupo N2F

Docente: Luis Dentone

Fecha de entrega del documento 04-05-2023

TABLA DE CONTENIDO

DIAGRAMA DE CLASES - UML



INFORME DE TESTING

CASOS DE PRUEBA PARA DATOS PRECARGADOS CORRESPONDIENTES A HUESPED, OPERADOR, ACTIVIDAD_HOSTEL Y ACTIVIDAD_TERCERIZADA.

Datos Huésped:

ESCENARIO DE TEST	DATOS UTILIZADOS	RESULTADO ESPERADO	RESULTADO OBTENIDO	ESTADO (F=FALLA, P=PASA)
Ingreso de datos precargados a la aplicación	"ci", "46112142", "Alvaro", "Perez", "7", " 2", fecha, "Huesped2@ort", "11223344"	Precarga cumple con la estructura de la clase – El huésped es ingresado.	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"ci", "46112142", "", "Perez", "7", " 2", fecha, "Huesped2@ort", "11223344" //Nombre String vacío	Se arroja excepción indicando que se deben completar todos los campos, huésped no ingresado.	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"ci", "46112142", "Alvaro", "", "7", " 2", fecha, "Huesped2@ort", "11223344" //Apellido String vacío	Se arroja excepción indicando que se deben completar todos los campos, huésped no ingresado.	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"credencial", "46112142", "Alvaro", "Perez", "7", " 2", fecha, "Huesped2@ort", "11223344" //Tipo documento Credencial	Se arroja excepción indicando que se deben completar todos los campos, huésped no ingresado. //Debe ingresar ci, pasaporte y otros	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"", "46112142", "Alvaro", "Perez", "7", " 2", fecha, "Huesped2@ort", "11223344" //Tipo documento string vacio	Se arroja excepción debe completar todos los campos	Resultado esperado	P

Ingreso de datos precargados a la aplicación	"ci", "46112148", "Alvaro", "Perez", "7", " 2", fecha, "Huesped2@ort", "11223344" //ci que no existe digito verificador 8	Se arroja excepcion. La cedula no es valida	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"ci", "46112142", "Alvaro", "Perez", "", " 2", fecha, "Huesped2@ort", "11223344" //Habitacion string vacio	Se arroja excepcion. Debe completar todos los campos	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"ci", "46112142", "Alvaro", "Perez", "7", " 2", fecha, "Huesped2@ort", "11223344" //Fidelizacion string vaci	Se arroja excepcion. Debe completar todos los campos	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"ci", "46112142", "Alvaro", "Perez", "7", " 5", fecha, "Huesped2@ort", "11223344" //Fidelizacion numero que no esta entre 1 y 4 (5)	Se arroja excepcion. El numero de fidelizacion debe estar entre 1 y 4	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"ci", "46112142", "Alvaro", "Perez", "7", " 2", fecha, "Huesped2@ort", "11223344" //Fecha de nacimiento posterior al dia de hoy	Se arroja excepcion. La fecha de nacimiento no puede ser mayor a la fecha actual	Resultado esperado	P

Datos Operador:

ESCENARIO DE TEST	DATOS UTILIZADOS	RESULTADO ESPERADO	RESULTADO OBTENIDO	ESTADO (F=FALLA, P=PASA)
Ingreso de datos precargados a la aplicación	"roberto@gmail.com", "11223344" //Ingreso datos correctos	Precarga cumple con la estructura de la clase – El operador es ingresado.	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"roberto@gmail.com", "11223344" //Email string vacio	Se arroja excepcion. La direccion de correo electronica no es valida	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"robertogmail.com", "11223344" //Email sin arroba	Se arroja excepcion. La direccion de correo electronica no es valida	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"robertogmail.com@", "11223344" //Email con arroba al final	Se arroja excepcion. La direccion de correo electronica no es valida	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"@robertogmail.com", "11223344" //Email con arroba al principio	Se arroja excepcion. La direccion de correo electronica no es valida	Resultado esperado	P

Ingreso de datos precargados a la aplicación	"@robertogmail.com@", "11223344" //Email con arroba al principio y al final	Se arroja excepcion. La direccion de correo electronica no es valida	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"@robertogmail.com", "" //Contrasenia string vacio	Se arroja excepcion.La contrasenia debe tener al menos 8 caracteres	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"@robertogmail.com", "1122334" //Contrasenia con siete caracteres	Se arroja excepcion.La contrasenia debe tener al menos 8 caracteres	Resultado esperado	P

Datos Proveedor:

ESCENARIO DE TEST	DATOS UTILIZADOS	RESULTADO ESPERADO	RESULTADO OBTENIDO	ESTADO (F=FALLA, P=PASA)
Ingreso de datos precargados a la aplicación	"DreamWorks S.R.L.", "23048549", "Suarez 3380 Apto 304", "10" //Ingreso datos correctos	Precarga cumple con la estructura de la clase – El proveedor es ingresado.	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"", "23048549", "Suarez 3380 Apto 304", "10" //Nombre String vacío	Se arroja excepcion. Debe completar todos los campos	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"DreamWorks S.R.L.", "", "Suarez 3380 Apto 304", "10" //Ingreso datos correctos	Se arroja excepcion. Debe completar todos los campos	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"DreamWorks S.R.L.", "23048549", "", "10" //Direccion string vacio	Se arroja excepcion. Debe completar todos los campos	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"DreamWorks S.R.L.", "23048549", "Suarez 3380 Apto 304", "" //Descuento string vacío	Se arroja excepcion. Debe completar todos los campos	Resultado esperado	P

Datos Actividad

ESCENARIO DE TEST	DATOS UTILIZADOS	RESULTADO ESPERADO	RESULTADO OBTENIDO	ESTADO (F=FALLA, P=PASA)
Ingreso de datos precargados a la aplicación	"Cabalgata", "Cabalgata a la tarde", fecha, 15, 12, "Actividad_Hostel", "Luis Dentone", "Pradera del Hostel", true, 300 //Ingreso datos correctos	Precarga cumple con la estructura de la clase – La actividad es ingresada.	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"", "Cabalgata a la tarde", fecha, 15, 12, "Actividad_Hostel", "Luis Dentone", "Pradera del Hostel", true, 300 //Nombre string vacio	Se arroja excepcion. El nombr de la actividad no puede ser vacio	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"Cabalgata", "", fecha, 15, 12, "Actividad_Hostel", "Luis Dentone", "Pradera del Hostel", true, 300 //Descripcion Ingreso vacio	Se arroja excepcion. La descripcion de la actividad no puede ser vacia	Resultado esperado	P

Ingreso de datos precargados a la aplicación	"Cabalgata", "Cabalgata a la tarde", fecha, 15, 12, "Actividad_Hostel", "Luis Dentone", "Pradera del Hostel", true, 300 //Ingreso de fecha menor al dia de hoy	Se arroja excepcion . La fecha no puede ser menor a la fecha actual	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"Cabalgata", "Cabalgata a la tarde", fecha, 15, 12, "Actividad_Hostel", "Luis Dentone", "Pradera del Hostel", true //Sin Costo ingresado	Precarga cumple con la estructura de la clase – La actividad es ingresada. El costo es igual a cero	Resultado esperado	P
Ingreso de datos precargados a la aplicación	"Cabalgata", "Cabalgata a la tarde", fecha, 15, 12, "Actividad_Hostel", "Luis Dentone", "Pradera del Hostel", true //Sin Costo ingresado	Precarga cumple con la estructura de la clase – La actividad es ingresada. El costo es igual a cero	Resultado esperado	P

CÓDIGO

CONSOLA

PROGRAM

```
using Dominio;
using System.Globalization;
using System.Runtime.CompilerServices;
using System.Security.Cryptography.X509Certificates;

namespace Consola
{
    internal class Program
    {
        // Creación de Sistema mediante Instancia siguiendo el patrón Singleton
        static private Sistema _sistema = Sistema.Instancia;

        static void Main(string[] args)
        {
            // Try y catch para controlar validaciones con la precarga.
            try
            {
                _sistema.Precargar();
                Iniciar();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }

        //Método para mostrar el menú y acceder a sus diferentes métodos.
        static void Iniciar()
        {
            int opcion = 1;
            do
            {
                if (opcion != 0)
                {
                    MostrarMenu("Seleccionar (del 1 al 6):\n1-Listar Actividades\n2-
Listar Proveedores Alfabeticamente\n3-Listar Actividad Segun Fecha y Costo\n" +
                        "4-Establecer Valor de Promoción\n5-Alta de Huespedes\n6-Mostrar
Agenda\n" +
                        "0-Salir");
                    try
                    {
                        opcion = PedirNumero();
                        switch (opcion)
                        {
```

```
                    case 1:
                        ListarActividades();
                        break;
                    case 2:
                        ListarProveedores();
                        break;
                    case 3:
                        ActividadSegunFechaYCosto();
                        break;
                    case 4:
                        ValorDePromocion();
                        break;
                    case 5:
                        AltaDeHuespedes();
                        break;
                    case 6:
                        MostrarAgenda();
                        break;
                    default:
                        break;
                }
            }
            catch (Exception)
            {
                Console.WriteLine("Ingrese una opción válida\n");
            }

        }
    } while (opcion != 0);

    MostrarFinPrograma();
}

//Método que muestra el texto del menú
static void MostrarMenu(string texto)
{
    Console.WriteLine(texto);
}

//Metodo para mostrar el final del programa
static void MostrarFinPrograma()
{
    Console.WriteLine("Programa Finalizado");
}

//Método para pedir número de acuerdo al switch del menú
static int PedirNumero()
{
    Console.WriteLine("Ingrese un numero entre 1 y 6 o cero para terminar");
    int numero = int.Parse(Console.ReadLine());
    ControlarNumero(numero);
    return numero;
}
```

```
        //Método para controlar el ingreso de los números del menú
        static void ControlarNumero(int numero)
        {
            if (numero > 6 || numero < 0)
            {
                MensajeErrorNumeros();
            }
        }

        //Método para mostrar el error en el ingreso de los números del menú
        static void MensajeErrorNumeros()
        {
            Console.WriteLine("Debe ingresar un numero entre 1 y 6\n");
        }

        //METODOS DEL MENÚ

        //Método Listar Actividades
        static void ListarActividades()
        {
            Console.WriteLine("Listado de Actividades: ");


            foreach (Actividad item in _sistema.Actividades)
            {
                Console.WriteLine(item);
            }
        }

        //Metodo para listar proveedores ordenados usando la lista ordenada creada en
Sistema.
        static void ListarProveedores()
        {

            Console.WriteLine("Listado de Proveedores: ");

            foreach (Proveedor item in _sistema.ProveedoresOrdenados())
            {
                Console.WriteLine("-" + item);

            }

        }

        //Metodo para listar Actividad según fecha y costo, usando la lista ordenada
creada en Sistema.
        static void ActividadSegunFechaYCosto()
        {
            Console.WriteLine("LISTADO SEGUN FECHA Y COSTO\n");

            DateTime fecha1;
            DateTime fecha2;
            decimal costo;
```

```
            try
            {
                fecha1 = Utilidades.PedirFechaActividad();
                fecha2 = Utilidades.PedirFechaActividadHasta(fecha1);
                costo = Utilidades.PedirCosto();

                foreach (Actividad item in _sistema.ListarActividadesFechaCosto(fecha1,
fecha2, costo))
                {
                    Console.WriteLine("Costo de la actividad: " + item.Costo);
                    Console.WriteLine(item);
                }
            }
            catch (Exception)
            {
                return;
            }

        }

        //Método para establecer valor de promoción de un proveedor seleccionada
        static void ValorDePromocion()
        {
            Proveedor proveedor = SeleccionarProveedor();

            string nuevoDescuento = Utilidades.PedirNuevoValorDescuento();
            int result;
            if (int.TryParse(nuevoDescuento, out result))
            {
                proveedor.Descuento = nuevoDescuento;
                Console.WriteLine("El valor se modifico correctamente\n");
            }
            else
            {
                Console.WriteLine("El valor ingresado no es valido");
            }
        }

        //Método para seleccionar el proveedor por ID.
        static Proveedor SeleccionarProveedor()
        {
            Proveedor proveedor = null;
            ListarProveedores();
            do
            {
                try
                {
                    Console.WriteLine("Seleccione el proveedor al que desea modificarle
su valor promocional por el numero de ID\n");
                    int opcion = int.Parse(Console.ReadLine());
```

```
                    proveedor = _sistema.ObtenerProveedorPorId(opcion);

                    if (proveedor == null)
                    {
                        Console.WriteLine("El proveedor seleccionado no existe - Ingrese
Nuevamente ");

                    }
                }
                catch (Exception)
                {
                    Console.WriteLine("La opcion ingresada no es valida");
                }
            }
            while (proveedor == null);

            Console.WriteLine("Proovedor seleccionado:\n" + proveedor);

            return proveedor;
        }

        //Método para dar de Alta un Huésped con validación línea a línea.
        static void AltaDeHuespedes()
        {
            Console.WriteLine("ALTA DE HUESPED\n");
            string tipoDocumento = "";
            string numeroDocumento = "";
            string nombre = "";
            string apellido = "";
            string habitacion = "";
            string fidelizacion = "";
            DateTime fechaDeNacimiento;
            string email = "";
            string contrasenia = "";

            try
            {
                tipoDocumento = Utilidades.PedirTipoDocumento(); //Validacion linea a
linea
                numeroDocumento = Utilidades.PedirNumeroDocumento(tipoDocumento);
                ValidarNumeroDocNoRepetido(numeroDocumento);
                nombre = Utilidades.PedirStringAlta("Ingrese el nombre");
                apellido = Utilidades.PedirStringAlta("Ingrese el apellido");
                habitacion = Utilidades.PedirHabitacionAlta();
                fidelizacion = Utilidades.PedirFidelizacion();
                fechaDeNacimiento = Utilidades.PedirFechaNacimiento();
                email = Utilidades.PedirEmail();
                contrasenia = Utilidades.PedirContrasenia();

                Usuario unHuesped = new Huesped(tipoDocumento, numeroDocumento, nombre,
apellido, habitacion, fidelizacion, fechaDeNacimiento, email, contrasenia);

                try
```

```
                {
                    Console.WriteLine("Ingresando...\n");
                    _sistema.AgregarUsuario(unHuesped); // Aqui se valida el huésped con
las validaciones de la clase, que están en el
método AgregarUsuario() en Sistema
                    Console.WriteLine("El huesped ha sido registrado correctamente\n");
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message);
                }

            }
            catch (Exception)
            {
                return;
            }
        }

        //Método que valida la unicidad del documento, en la validación línea a línea
del Alta de Huésped
        static void ValidarNumeroDocNoRepetido(string numeroDocumento)
        {
            if (_sistema.ObtenerHuespedPorCedula(numeroDocumento) != null)
            {
                Console.WriteLine("Ya existe un usuario con ese documento\n");
                throw new Exception("Ya existe un usuario con ese documento");
            }
        }

        //Método para mostrar la Agenda.
        static void MostrarAgenda()
        {
            Console.WriteLine("Agenda: ");
            int numeroAgenda = 1;

            foreach (Agenda item in _sistema.Agenda)
            {
                Console.WriteLine(numeroAgenda + "-" + item);
                numeroAgenda++;
            }
        }

    }
}
```

UTILIDADES

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Net.NetworkInformation;
using System.Reflection.Metadata;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace Consola
{
    //Creación de la clase Utilidades para auxiliar en las validaciones y mensajes mostrados en Consola. Todos los métodos finalizan si se pulsa 0, retornando al menú principal
    public static class Utilidades
    {
        //Método para perdir string, y validar que no sea nula o vacía.
        public static string PedirString(string mensaje)
        {
            string textoPedido;
            do
            {
                Console.WriteLine(mensaje);
                textoPedido = Console.ReadLine();
                if (string.IsNullOrEmpty(textoPedido))
                {
                    Console.WriteLine("Debe ingresar el dato requerido");
                }

            } while (string.IsNullOrEmpty(textoPedido));
            return textoPedido;
        }

        //Méotodo para pedir la fecha "Desde", validarla, y mostrar los mensajes según corresponda.
        public static DateTime PedirFechaActividad() {

            string mensaje = "Ingrese fecha desde en el siguiente formato yyyy/MM/dd";
            int number;
            string fecha = PedirString(mensaje);
            if (fecha == "0")
            {
                throw new Exception();
            }
            bool isNumber = int.TryParse(fecha, out number);
            DateTime fechaConvertida = DateTime.MinValue;
            try
            {
                //Convertir el string a fecha
```

```
                fechaConvertida = DateTime.ParseExact(fecha, "yyyy/MM/dd",
CultureInfo.InvariantCulture);
            }

            catch (Exception)
            {
                Console.WriteLine("La fecha ingresada no es valida");
                PedirFechaActividad();
            }
            return fechaConvertida;
        }

        //Método para pedir la fecha "Hasta", validarla internamente y de acuerdo a la
fecha "Desde", y mostrar los mensajes según corresponda.
        public static DateTime PedirFechaActividadHasta(DateTime fecha)
        {

            string mensaje = "Ingrese fecha hasta en el siguiente formato yyyy/MM/dd";
            int number;
            string fechaHasta = PedirString(mensaje);
            if (fechaHasta == "0")
            {
                throw new Exception();
            }
            bool isNumber = int.TryParse(fechaHasta, out number);
            DateTime fechaConvertida = DateTime.MinValue;
            try
            {   //Convierte el string a fecha
                fechaConvertida = DateTime.ParseExact(fechaHasta, "yyyy/MM/dd",
CultureInfo.InvariantCulture);
                if (fechaConvertida < fecha)
                {
                    Console.WriteLine("Fecha desde no puede ser menor que fecha hasta");
                    PedirFechaActividad();
                }
                return fechaConvertida;
            }

            catch (Exception)
            {
                Console.WriteLine("La fecha ingresada no es valida");
                return PedirFechaActividadHasta(fecha);
            }
        }

        //Método para pedir costo, y validar que se ingresen números.
        public static decimal PedirCosto()
        {
            string mensaje = "Ingrese un costo (se mostraran todas las actividades con
costo mayor al ingresado, en forma descendente)\n";
            string costo = PedirString(mensaje);
            decimal numero;
            bool esNumero = decimal.TryParse(costo, out numero);
```

```
            if (costo == "0")
            {
                throw new Exception();
            }
             if(esNumero == false)
            {
                Console.WriteLine("Debe ingresar numeros");
                return PedirCosto();
            }
            decimal costoConvertido = int.Parse(costo);
            return costoConvertido;
        }

        //Método para pedir Tipo de Documento y garantizar que se ingrese el dato
solicitado.
        public static string PedirTipoDocumento()
        {
            string mensaje = "Ingrese el tipo de Documento (El documento solo puede ser
: CI, Pasaporte u Otros. 0 para salir)";

            string tipoDocumento = PedirString(mensaje);
            tipoDocumento = tipoDocumento.ToLower();
            if (tipoDocumento == "0")
            {
                throw new Exception();
            }
            if (tipoDocumento == "ci" || tipoDocumento == "pasaporte" || tipoDocumento
== "otros")
            { return tipoDocumento; }

            return PedirTipoDocumento();
        }

        //Método para pedir Número de documento, validarlo, y mostrar los mensajes según
corresponda.
        public static string PedirNumeroDocumento(string tipoDocumento)
        {
            string mensaje = "Ingrese el numero de Documento. 0 para salir";
            string numeroDocumento = PedirString(mensaje);
            int numero;
            bool esNumero = int.TryParse(numeroDocumento, out numero);

            if (numeroDocumento == "0")
            {
                throw new Exception();
            }
            numeroDocumento = numeroDocumento.ToLower();

            if (tipoDocumento == "ci")
            {   //Validación de la longitud de la cédula
                if (numeroDocumento.Length != 8)
                {
                    Console.WriteLine("La cédula debe tener 8 dìgitos y ser solo
```

```
numeros");
                    return PedirNumeroDocumento(tipoDocumento);
                }

                if (!esNumero)
                {
                    return PedirNumeroDocumento(tipoDocumento);
                }

                if (numeroDocumento.Length == 8) //Validación de la cédula uruguaya
                {
                    int suma = 0;

                    for (int i = 0; i < 7; i++)
                    {
                        suma += (int.Parse("2987634"[i].ToString()) *
int.Parse(numeroDocumento[i].ToString())) % 10;
                    }

                    int digitoVerificador = 0;

                    if (suma % 10 != 0)
                    {
                        digitoVerificador = 10 - suma % 10;
                    }

                    int ultimoDigito = int.Parse(numeroDocumento[numeroDocumento.Length
- 1].ToString());
                    if (ultimoDigito != digitoVerificador)
                    {
                        Console.WriteLine("La cédula no es válida");
                        return PedirNumeroDocumento(tipoDocumento);

                    }
                    return numeroDocumento;
                }

            } //Validación de Pasaporte y Otros
            if (tipoDocumento == "pasaporte" || tipoDocumento == "otros")
            {
                int number;
                bool isNumber = int.TryParse(numeroDocumento, out number);
                if (isNumber)
                {
                    return numeroDocumento;
                }
                else {
                    Console.WriteLine("Debe ingresar numeros");
                    return PedirNumeroDocumento(tipoDocumento);
                }

            }
            return PedirNumeroDocumento(tipoDocumento);
```

```
        }

        //Método para pedir Nombre y Apellido en Alta de huésped, y validar que nos sean
vacíos.
        public static string PedirStringAlta(string mensaje)
        {
            string textoPedido;

            do
            {
                Console.WriteLine(mensaje);
                textoPedido = Console.ReadLine();
                if (textoPedido == "0")
                {
                    throw new Exception();
                }
                if (string.IsNullOrEmpty(textoPedido))
                {
                    Console.WriteLine("Debe ingresar el dato requerido");
                }

            } while (string.IsNullOrEmpty(textoPedido));
            return textoPedido;
        }

        //Método paa pedir habitación, validar que se ingrese un número y que no sea
vacío.
        public static string PedirHabitacionAlta()
        {
            int number;
            string mensaje = "Ingrese el numero de habitacion";
            string habitacion = PedirString(mensaje); ;
            bool isNumber = int.TryParse(habitacion, out number);


            if (habitacion == "0")
            {
                throw new Exception();
            }
            if (string.IsNullOrEmpty(habitacion))
            {
                Console.WriteLine("Debe ingresar el dato requerido");
            }
            if (!isNumber)
            {
                Console.WriteLine("Debe ingresar numeros");
                return PedirHabitacionAlta();
            }
            return habitacion;
        }

        //Método para pedir fidelización, validar que sea un número del 1 al 4 y no sea
vacío
```

```
        public static string PedirFidelizacion()
        {
            string fidelizacion = "";
            string mensaje = "Ingrese fidelización (1 al 4)";
            int number;
            fidelizacion = PedirString(mensaje);
            bool isNumber = int.TryParse(fidelizacion, out number);

            if (fidelizacion == "0")
            {
                throw new Exception();
            }
            if (!isNumber)
            {
                Console.WriteLine("Debe ingresar numeros");
                return PedirFidelizacion();
            }
            if (int.Parse(fidelizacion) < 1 || int.Parse(fidelizacion) > 4)
                {
                    Console.WriteLine("Fidelizacion debe ser un numero del 1 al 4");
                    return PedirFidelizacion();
                }
            return fidelizacion;
        }

        //Método para pedir Fecha de nacimiento, validar que tenga el formato correcto y
que no sea posterior a la fecha actual.
        public static DateTime PedirFechaNacimiento()
        {
            string mensaje = "Ingrese la fecha de nacimiento en el siguiente formato
yyyy/MM/dd";
            int number;
            string fechaNacimiento = PedirString(mensaje);
            if (fechaNacimiento == "0")
            {
                throw new Exception();
            }
            bool isNumber = int.TryParse(fechaNacimiento, out number);
            DateTime fechaConvertida = DateTime.MinValue;
            try
            {
                fechaConvertida = DateTime.ParseExact(fechaNacimiento, "yyyy/MM/dd",
CultureInfo.InvariantCulture);
                if (fechaConvertida > DateTime.Today)
                {
                    Console.WriteLine("La fecha de nacimiento debe ser posterior a la
fecha de hoy");
                    PedirFechaNacimiento();
                }
            }
            catch (Exception)
            {
                Console.WriteLine("La fecha ingresada no es valida");
```

```
                PedirFechaNacimiento();
            }
            return fechaConvertida;
        }

        //Métddo para pedir nuevo valor de promoción y validar que sea un número entre 0
y 100.
        public static string PedirNuevoValorDescuento()
        {
            int nuevoValorPromocion = -1;
            do
            {
                try
                {
                    Console.WriteLine("Ingrese el nuevo valor de promocion (entre 0 y
100)\n");
                    nuevoValorPromocion = int.Parse(Console.ReadLine());

                    if (nuevoValorPromocion < 0 || nuevoValorPromocion > 100)
                    {
                        Console.WriteLine("El Valor ingresado debe ser entre 0 y 100 ");
                    }

                }
                catch (Exception)
                {
                    Console.WriteLine("El Valor ingresado debe ser un numero");
                }
            }
            while (nuevoValorPromocion < 0 || nuevoValorPromocion > 100);
            return Convert.ToString(nuevoValorPromocion);
        }

        // Método para pedir el Email, validarlo y mostrar los mensajes según
corresponda.
        public static string PedirEmail()
        {
            string mensaje = "Ingrese el Email";

            string email = PedirString(mensaje).ToLower();

            if (email == "0")
            {
                throw new Exception();
            }

            if (email.IndexOf('@') < 1 || email.LastIndexOf('@') == email.Length - 1) {
                 Console.WriteLine("El email ingresado no es valido");
                PedirEmail();
            }
            return email;
        }
        // Método para pedir contraseña, validarla y mostrar los mensajes según
```

```
corresponda.
        public static string PedirContrasenia()
        {
            string mensaje = "Ingrese la contraseña";

            string contrasenia = PedirString(mensaje);
            if (contrasenia == "0")
            {
                throw new Exception();
            }

             if (contrasenia.Length < 8) {
                 Console.WriteLine("Contraseña muy corta, debe contener al menos 8
caracteres");
                 PedirContrasenia();
            }
            return contrasenia;
        }


    }

}
```

DOMINIO

ACTIVIDAD

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace Dominio

{

    //Creación de la clase Actividad, con la interfaz IValidable para garantizar la
existencia del metodo Validar(), la interfaz IEquatable para definir el Contain por ID,
e Icomparable para ordenarlas por Costo, en forma descendente
    public abstract class Actividad : IValidable, IEquatable<Actividad>,
IComparable<Actividad>
    {
        public int _id;
        public string NombreActividad { get; set; }
        public string Descripcion { get; set; }
        public DateTime FechaActividad { get; set; }
        public int CantidadMaximaParticipantes { get; set; }
```

```
        public int EdadMinima { get; set; }

        public string TipoDeActividad { get; set; }

        public static int ultIdActividad;
        public decimal Costo { get; set; }


        //Constructor
        public Actividad(string nombreActividad, string descripcion, DateTime
fechaActividad, int cantidadMaximaParticipantes, int edadMinima, string tipoDeActividad,
decimal costo)
        {
            NombreActividad = nombreActividad;
            Descripcion = descripcion;
            FechaActividad = fechaActividad;
            CantidadMaximaParticipantes = cantidadMaximaParticipantes;
            EdadMinima = edadMinima;
            TipoDeActividad = tipoDeActividad;
            Costo = costo;
            _id = ++ultIdActividad; //ID autogenerado
        }

        //Método Validar
        public void Validar()
        {
            ValidarFechaActividad();
            ValidarNombreActividad();
            ValidarDescripcionActividad();
            ValidarCostoActividad();
        }

        //Método para validar nombre de la Actividad según los requerimientos
        public void ValidarNombreActividad()
        {
            if (string.IsNullOrEmpty(NombreActividad))
            {
                throw new Exception("El Nombre de la actividad no puede ser vacìo");
            }

            else if (NombreActividad.Length > 25)
            {
                throw new Exception("El Nombre no puede tener mas de 25 caracteres");
            }
        }

        //Método para validar que la descripción de la actividad no sea nula o vacía.
        public void ValidarDescripcionActividad()
        {
            if (string.IsNullOrEmpty(Descripcion))
            {
                throw new Exception("La descripción de la actividad no puede ser
vacìa");
```

```
            }
        }

        //Método para validar que la fecha de la actividad no sea menor que la fecha
actual
        public void ValidarFechaActividad()
        {
            if (FechaActividad < DateTime.Now)
            {
                throw new Exception("La fecha de la actividad no puede ser menor a la
fecha actual");
            }
        }

        //Método para garantizar que sino recibe costo, el mismo sea 0.
        public void ValidarCostoActividad()
        {
            if (Costo == null) Costo = 0;
        }

        // Definición del Contains por ID
        public bool Equals(Actividad? other)
        {
            return other != null && _id == other._id;
        }

        //Ordena las actividades en orden descendente
        public int CompareTo(Actividad? other)
        {
            if (other == null)
                return 0;
            return -1 * Costo.CompareTo(other.Costo);
        }

    }
}
```

ACTIVIDAD_HOSTEL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace Dominio
{

    //Creación de la clase Actividad_Hostel, con herencia de la clase Actividad, y la
interfaz IValidable para garantizar la existencia del método Validar.
    public class Actividad_Hostel : Actividad, IValidable
```

```
    {
        public string ResponsableActividad { get; set; }
        public string LugarDentroHostal { get; set; }
        public bool AireLibre { get; set; }

        //Constructor con herencia
        public Actividad_Hostel(string nombreActividad, string descripcion, DateTime
fechaActividad,
            int cantidadMaximaParticipantes, int edadMinima, string tipoDeActividad,
            string responsableActividad, string lugarDentroHostal, bool aireLibre,
decimal costo = 0)
            :base(nombreActividad, descripcion, fechaActividad,
cantidadMaximaParticipantes, edadMinima,
            tipoDeActividad, costo)
        {
            ResponsableActividad = responsableActividad;
            LugarDentroHostal = lugarDentroHostal;
            AireLibre = aireLibre;
        }

        //Método Validar
        public void Validar()
        {
            ValidarResponsable();
            ValidarLugar();
        }

        //Método para validar que el responsable no sea nulo o vacio
        public void ValidarResponsable()
        {
            if (string.IsNullOrEmpty(ResponsableActividad))
            {
                throw new Exception("El Nombre del responsable de la actividad no puede
ser vacìo");
            }
        }

        //Método para validar que el Lugar no sea nulo o vacio
        public void ValidarLugar()
        {
            if (string.IsNullOrEmpty(LugarDentroHostal))
            {
                throw new Exception("El Nombre del responsable de la actividad no puede
ser vacìo");
            }
        }

        //Sobreescritura del método ToString() para mostrar la información según los
requerimientos.
        public override string ToString()
        {
            string respuesta = string.Empty;
            respuesta += $"ID actividad: {_id}\n";
            respuesta += $"Nombre de Actividad: {NombreActividad}\n";
            respuesta += $"Descripcion: {Descripcion}\n";
```

```
            respuesta += $"Fecha de la actividad: {FechaActividad}\n";
            respuesta += $"Cantidad Máxima de Participantes:
{CantidadMaximaParticipantes}\n";
            respuesta += $"Edad Mínima para participar: {EdadMinima}\n";
            respuesta += $"Responsable: {ResponsableActividad}\n";
            respuesta += $"Costo: {Costo}\n";
            return respuesta;

        }
    }
}
```

ACTIVIDAD_TERCERIZADA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Dominio
{

    //Creación de la clase Actividad_Tercerizada, con herencia de la clase Actividad, y
la interfaz IValidable para garantizar la existencia del método Validar.
    public class Actividad_Tercerizada : Actividad, IValidable
    {
        public Proveedor Proveedor { get; set; }
        public bool ActividadConfirmada { get; set; }
        public DateTime FechaDeConfirmacion { get; set; }

        //Constructor
        public Actividad_Tercerizada(string nombreActividad, string descripcion,
            DateTime fechaActividad, int cantidadMaximaParticipantes,
            int edadMinima, string tipoDeActividad, Proveedor proveedor,
            bool actividadConfirmada, DateTime fechaConfirmacion, decimal costo = 0)
:base(nombreActividad, descripcion,
            fechaActividad, cantidadMaximaParticipantes, edadMinima, tipoDeActividad,
costo)
        {
            Proveedor = proveedor;
            ActividadConfirmada = actividadConfirmada;
            FechaDeConfirmacion = fechaConfirmacion;
        }

        //Método Validar
        public void Validar()
        {
            ValidarFechaConfirmacion();
        }
```

```
        //Validar que fecha de confirmacion no sea nula
        public void ValidarFechaConfirmacion()
        {
            if (FechaDeConfirmacion==null)
            {
                throw new Exception("La fecha de confirmacion no puede ser vacia");
            }
        }

        //ToString para mostrar la actividad de acuerdo con los requerimientos.
        public override string ToString()
        {
            string respuesta = string.Empty;
            respuesta += $"ID actividad: {_id}\n";
            respuesta += $"Nombre de Actividad: {NombreActividad}\n";
            respuesta += $"Descripcion: {Descripcion}\n";
            respuesta += $"Fecha de la actividad: {FechaActividad}\n";
            respuesta += $"Cantidad Máxima de Participantes:
{CantidadMaximaParticipantes}\n";
            respuesta += $"Edad Mínima para participar: {EdadMinima}\n";
            respuesta += $"Proveedor: {Proveedor.Nombre}\n";
            respuesta += $"Costo: {Costo}\n";
            return respuesta;
        }
    }
}
```

AGENDA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace Dominio
{

    //Creación de la clase Agenda, con la interfaz IValidable para garantizar la
existencia del método Validar().
    public class Agenda: IValidable
    {
        public Huesped Huesped { get; set; }
        public Actividad Actividad { get; set; }
        public string EstadoAgenda { get; set; }

        //Constructor
        public Agenda(Huesped huesped, Actividad actividad, string estadoAgenda)
        {
            Huesped = huesped;
            Actividad = actividad;
            EstadoAgenda = estadoAgenda;
```

```
        }

        public void Validar()
        {
        }

        //ToString() para mostrar la agenda.
        public override string ToString()
        {
            string respuesta = string.Empty;
            respuesta += $"Nombre Huésped: {Huesped.Nombre} {Huesped.Apellido} \n";
            respuesta += $"Nombre Actividad:  {Actividad.NombreActividad}\n";
            respuesta += $"Fecha Actividad:  {Actividad.FechaActividad}\n";
            respuesta += $"Estado: {EstadoAgenda}\n";
            return respuesta;
        }
    }
}
```

HUESPED

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace Dominio
{

    //Creación de la clase Huèsped, con herencia de Usuario, la interfaz IValidable para
garantizar la existencia del metodo Validar() y la interfaz IEquatable para definir el
Contains por número de documento.
    public class Huesped : Usuario, IValidable, IEquatable<Huesped>
    {
        public string TipoDocumento { get; set; }
        public string NumeroDocumento { get; set; }
        public string Nombre { get; set; }
        public string Apellido { get; set; }
        public string Habitacion { get; set; }
        public string Fidelizacion { get; set; }

        public DateTime FechaNacimiento { get; set; }

        //Constructor con herencia
        public Huesped(string tipoDocumento, string numeroDocumento, string nombre,
            string apellido, string habitacion, string fidelizacion, DateTime
fechaNacimiento,
            string email, string contrasenia) :
            base(email, contrasenia)
        {
            TipoDocumento = tipoDocumento;
            NumeroDocumento = numeroDocumento;
```

```
            Nombre = nombre;
            Apellido = apellido;
            Habitacion = habitacion;
            Fidelizacion = fidelizacion;
            FechaNacimiento = fechaNacimiento;
        }

        //Métodos
        public void Validar()
        {
            ValidarNombre();
            ValidarApellido();
            ValidarTipoDocumento();
            ValidarNumeroDocumento();
            ValidarHabitacion();
            ValidarFidelizacion();
            ValidaFechaNacimiento();



        }

        //Método para validar String vacío
        public void ValidarStringVacio(string parametro)
        {
            if (string.IsNullOrEmpty(parametro))
            { throw new Exception("Debe completar todos los campos"); }
        }

        //Método para validar que Nombre no sea string vacío
        public void ValidarNombre()
        {
            ValidarStringVacio(Nombre);
        }

        //Método para validar que Apellido no sea string vacío
        public void ValidarApellido()
        {
            ValidarStringVacio(Apellido);
        }

        //Método para validar Tipo de Documento
        public void ValidarTipoDocumento()
        {
            ValidarStringVacio(TipoDocumento);
            TipoDocumento = TipoDocumento.ToLower();

            if (TipoDocumento != "ci" && TipoDocumento != "pasaporte" && TipoDocumento
!= "otros")
            {
                throw new Exception("El documento solo puede ser : CI, Pasaporte u
Otros");
            }
        }
```

```
        //Método para validar Número de Documento según los requerimientos y el dígito
verificador de la cédula uruguaya.
        public void ValidarNumeroDocumento()
        {

            ValidarStringVacio(NumeroDocumento);

            TipoDocumento = TipoDocumento.ToLower();
            if (TipoDocumento == "ci")
            {
                if (NumeroDocumento.Length != 8)
                {
                    throw new Exception("La cédula debe tener 8 dìgitos");
                }
                else if (NumeroDocumento.Length == 8)
                {
                    int suma = 0;

                    for (int i = 0; i < 7; i++)
                    {
                        suma += (int.Parse("2987634"[i].ToString()) *
int.Parse(NumeroDocumento[i].ToString())) % 10;
                    }

                    int digitoVerificador = 0;

                    if (suma % 10 != 0)
                    {
                        digitoVerificador = 10 - suma % 10;
                    }

                    int ultimoDigito = int.Parse(NumeroDocumento[NumeroDocumento.Length
- 1].ToString());
                    if (ultimoDigito != digitoVerificador)
                    {
                        throw new Exception("La cédula no es válida");
                    }
                }
            }
        }

        //Método para validar que habitación no sea un string vacío, y sea un número
        public void ValidarHabitacion()
        {
            ValidarStringVacio(Habitacion);
            int number;
            bool isNumber = int.TryParse(Habitacion, out number);
            if (!isNumber) {
                throw new Exception("La habiración debe ser un número");
            }
        }

        public void ValidaFechaNacimiento()
        {
            if (FechaNacimiento > DateTime.Today)
```

```
            {
                throw new Exception("La fecha de nacimiento no puede ser mayor a la
fecha actual");
            }
        }

        //Método para que fidalización no sea un string vacío, y sea un número del 1 al
4
        public void ValidarFidelizacion()
        {
            ValidarStringVacio(Fidelizacion);
            int fidel = int.Parse(Fidelizacion);

            if (fidel > 4 || fidel < 1)
            {
                throw new Exception("El numero de Fidelizacion debe estar entre 1 y 4");
            }
        }

        //Sobrescritura del Contains para que verifique unicidad por Tipo y Número de
Documento
        public bool Equals(Huesped? other)
        {
            return other != null && (TipoDocumento == other.TipoDocumento) &&
(NumeroDocumento == other.NumeroDocumento);
        }
    }
}
```

IVALIDABLE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Dominio
{
    //Ivalidable para definir el contrato
    internal interface IValidable
    {
        public void Validar();
    }
}
```

OPERADOR

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Dominio

{
    //Creación de la clase Operador, con herencia de Usuario, la interfaz IValidable
para garantizar la existencia del método Validar() y la interfaz IEquatable para definir
el Contains por Email.
    public class Operador : Usuario, IEquatable<Operador>, IValidable
    {


        public Operador(string email, string contrasenia) :
            base(email, contrasenia)
        {

        }

        //Método Validar
        public void Validar()
        {
            validarEmail();
            validarContrasenia();
        }

        //Método para validar el Email de acuerdo con los requerimientos.
        public void validarEmail()
        {

            if (Email.Contains("@") && Email.IndexOf("@") > 0 && Email.LastIndexOf("@")
< Email.Length - 1)
            {
            }
            else
            {
                throw new Exception("La dirección de correo electrónico no es válida.");
            }
        }

        //Método para validar la contraseña de acuerdo con los requerimientos.
        public void validarContrasenia()
        {
            if (Contrasenia.Length < 8)
            {
                throw new Exception("La Contraseña debe contener al menos 8
caracteres");
            }
        }

        // Definición del Contains por Email
        public bool Equals(Operador? other)
        {
            return other != null && Email == other.Email;
```

```
        }
    }
}
```

PROVEEDOR

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Runtime.InteropServices.JavaScript.JSType;

namespace Dominio
{
    //Creación de la clase Proveedor, con Ivalidable para garantizar que tenga el método
Validar, IComparable para ordenarlos por nombre e IEquatable para definir el Contains
por Nombre
    public class Proveedor : IValidable, IComparable<Proveedor>, IEquatable<Proveedor>
    {

        public int _id;
        public string Nombre { get; set; }
        public string NumeroContacto { get; set; }
        public string Direccion { get; set; }
        public string Descuento { get; set; }

        public static int ultIdProveedor;

        //Constructor
        public Proveedor(string nombre, string numeroContacto, string direccion, string
descuento)
        {
            Nombre = nombre;
            NumeroContacto = numeroContacto;
            Direccion = direccion;
            Descuento = descuento;
            _id= ++ultIdProveedor;
        }

        //Método Validar
        public void Validar()
        {
            ValidarDireccion();
            ValidarNombre();
            ValidarNumeroContacto();
            ValidarDescuento();
        }
        public override string ToString()
        {
            string respuesta = string.Empty;
            respuesta += $"Nombre: {Nombre}\n";
            respuesta += $"Numero de Contacto: {NumeroContacto}\n";
```

```
        respuesta += $"Dirección: {Direccion}\n";
        respuesta += $"Descuento actividades promocionales: {Descuento}\n";
        respuesta += $"ID: {_id}\n";
        return respuesta;
    }

    //Método para validar que el nombre del Proveedor no sea vacío o nulo
    public void ValidarNombre()
    {
        if (string.IsNullOrEmpty(Nombre))
    { throw new Exception("Debe completar todos los campos"); }
    }

    //Método para validar que la dirección del Proveedor no sea vacía o nula
    public void ValidarDireccion()
        {
            if (string.IsNullOrEmpty(Direccion))
            { throw new Exception("Debe completar todos los campos"); }
        }

    //Método para validar que el número de contacto no sea vacío o nulo
    public void ValidarNumeroContacto()
    {
        if (string.IsNullOrEmpty(NumeroContacto))
        { throw new Exception("Debe completar todos los campos"); }

    }

    //Método para validar que el descuento de proveedor no sea vacío o nulo
    public void ValidarDescuento()
    {
        if (string.IsNullOrEmpty(Descuento))
        { throw new Exception("Debe completar todos los campos"); }
    }

    //Para ordenar los proveedores por Nombre
    public int CompareTo(Proveedor? other)
    {
        if (other == null)
            return 0;
        return Nombre.CompareTo(other.Nombre);
    }

    //Definición del Contains por unicidad de Nombre
    public bool Equals(Proveedor? other)
    {
        return other != null && Nombre == other.Nombre;
    }
}

}
```

SISTEMA

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Intrinsics.X86;
using System.Text;
using System.Threading.Tasks;

namespace Dominio
{
    public class Sistema
    {

        //Creaciòn de listas, todas privadas.
        private List<Usuario> _usuarios = new List<Usuario>();
        private List<Actividad> _actividades = new List<Actividad>();
        private List<Proveedor> _proveedores = new List<Proveedor>();
        private List<Agenda> _agenda = new List<Agenda>();

        //Patron Singleton
        private static Sistema _instancia = null;
        public static Sistema Instancia
        {
            get
            {
                if (_instancia == null)
                {
                    _instancia = new Sistema();
                }
                return _instancia;
            }
        }
        private Sistema()
        {
        }

        // Para acceder a las listas, pero no modificarlas.
        public List<Usuario> Usuarios
        {
            get { return _usuarios; }
        }
        public List<Actividad> Actividades
        {
            get { return _actividades; }
        }
        public List<Proveedor> Proveedores
        {
            get { return _proveedores; }
        }
        public List<Agenda> Agenda
        {
            get { return _agenda; }
        }

        // Precarga de datos
        public void Precargar()
        {
```

```
            PrecargarDatosUsuarios();
            PrecargarProveedores();
            PrecargarActividades();
            PrecargarAgenda();
        }

        //Precarga de dos huéspedes y un operador
        private void PrecargarDatosUsuarios()
        {
            //Dos huespedes
            DateTime fecha = new DateTime(1989, 01, 06);
            Usuario unHuesped1 = new Huesped("ci", "46112136", "Jonatan", "Miranda",
"1", "3", fecha, "Huesped1@ort", "11223344");
            AgregarUsuario(unHuesped1);

            DateTime fecha2 = new DateTime(1986, 11, 29);
            Usuario unHuesped2 = new Huesped("ci", "46112142", "Alvaro", "Perez", "7",
"4", fecha2, "Huesped2@ort", "11223344");
            AgregarUsuario(unHuesped2);

            //Un operador
            Usuario unOperador = new Operador("roberto@gmail.com", "11223344");
            AgregarUsuario(unOperador);

        }

        //Metodo para validar el ususario según su clase y validaciones internas,
validar la unicidad de Documento e Email y luego agregarlo a la lista
        public void AgregarUsuario(Usuario usuario)
        {
            if (usuario == null)
            {
                throw new Exception("El usuario recibido no es vàlido.");
            }
            if (usuario is Huesped) //Casteo hacia abajo para acceder a las validaciones
de Huésped.
            {
                Huesped huesped = (Huesped)usuario;
                huesped.Validar(); //Validación del Huésped
                if (ListaHuespedes().Contains(huesped)) // Validación unicidad del
Documento usando el Equals definido en Huésped
                { throw new Exception("Ya existe un Huésped con este documento\n"); }
                if (_usuarios.Contains(huesped))// Validacion unicidad del Email usando
el Equals definido en Usuario
                {
                    throw new Exception("Ya existe un usario con ese correo\n");
                }
                _usuarios.Add(huesped); // Agrega el huésped
            } else if (usuario is Operador) //Casteo hacia abajo para acceder a las
validaciones de Operador.
            {
                Operador operador = (Operador)usuario;
                operador.Validar(); // Validacion del Operador
                 if (_usuarios.Contains(operador))//  Validacion unicidad del Email
usando el Equals definido en Usuario
```

```
                {
                    throw new Exception("Ya existe un usario con ese correo");
                    }
                _usuarios.Add(operador); // Agrega el operador
            }
        }

        //Precarga de Proveedores según los requerimientos
        public void PrecargarProveedores()
        {
            Proveedor proveedor1 = new Proveedor("DreamWorks S.R.L.", "23048549",
"Suarez 3380 Apto 304", "10");
            AgregarProveedor(proveedor1);
            Proveedor proveedor2 = new Proveedor("Estela Umpierrez S.A.", "33459678",
"Lima 2456", "7");
            AgregarProveedor(proveedor2);
            Proveedor proveedor3 = new Proveedor("TravelFun", "29152020", "Misiones
1140", "9");
            AgregarProveedor(proveedor3);
            Proveedor proveedor4 = new Proveedor("Rekreation S.A.", "29162019", "Bacacay
1211", "11");
            AgregarProveedor(proveedor4);
            Proveedor proveedor5 = new Proveedor("Alonso & Umpierrez", "24051920", "18
de Julio 1956 Apto 4", "10");
            AgregarProveedor(proveedor5);
            Proveedor proveedor6 = new Proveedor("Electric Blue", "26018945", "Cooper
678", "5");
            AgregarProveedor(proveedor6);
            Proveedor proveedor7 = new Proveedor("Lúdica S.A.", "26142967", "Dublin
560", "4");
            AgregarProveedor(proveedor7);
            Proveedor proveedor8 = new Proveedor("Gimenez S.R.L.", "29001010", "Andes
1190", "7");
            AgregarProveedor(proveedor8);
            Proveedor proveedor9 = new Proveedor("Fernandez S.R.L", "22041120",
"Agraciada 2512 Apto. 1", "8");
            AgregarProveedor(proveedor9);
            Proveedor proveedor10 = new Proveedor("Norberto Molina", "22001189",
"Paraguay 2100", "9");
            AgregarProveedor(proveedor10);
        }

        //Metodo para validar el Proveedor, asegurarse que no sea nulo, y que su nombre
sea único.
        private void AgregarProveedor(Proveedor proveedor)
        {
            if (proveedor == null)
            {
                throw new Exception("El proveedor recibido no es vàlido.");
            }
            if (_proveedores.Contains(proveedor)) // Validación unicidad del Nombre de
Proveedor utilizando el Equals definido en Proveedor.
            {
                throw new Exception($"El proveedor {proveedor.Nombre} ya existe.");
            }
```

```
            proveedor.Validar();
            _proveedores.Add(proveedor);
        }

        //Precarga de Actividades
        private void PrecargarActividades()
        {
            //ACTIVIDADES HOSTEL

            //1
            DateTime fecha = new DateTime(2023, 06, 01);
            DateTime fechaConfirmacion = new DateTime(2023, 05, 25);
            Actividad actividad1 = new Actividad_Hostel("Cabalgata", "Cabalgata a la
tarde", fecha, 15, 12, "Actividad_Hostel", "Luis Dentone", "Pradera del Hostel", true);
// Actividad cargada sin costo para verificar que entra con costo 0.
            AgregarActividad(actividad1);
            //2
            DateTime fecha2 = new DateTime(2023, 06, 02);
            DateTime fechaConfirmacion2 = new DateTime(2023, 05, 25);
            Actividad actividad2 = new Actividad_Hostel("Piscina", "Actividades en
Piscinas climatizadas", fecha2, 20, 10, "Actividad_Hostel",
                "Ricardo Dentone", "Piscinas del Hostel", false, 400);
            AgregarActividad(actividad2);
            //3
            DateTime fecha3 = new DateTime(2023, 06, 03);
            DateTime fechaConfirmacion3 = new DateTime(2023, 05, 25);
            Actividad actividad3 = new Actividad_Hostel("Kayak", "Kayak en el Rio",
fecha3, 5, 18, "Actividad_Hostel",
                "Maximo Dentone", "Rio lindero al Hostel", true, 150);
            AgregarActividad(actividad3);
            //4
            DateTime fecha4 = new DateTime(2023, 06, 04);
            DateTime fechaConfirmacion4 = new DateTime(2023, 05, 25);
            Actividad actividad4 = new Actividad_Hostel("Kayak", "Kayak en el Rio",
fecha4, 5, 18, "Actividad_Hostel",
                "Fernando Dentone", "Rio lindero al Hostel", true, 100);
            AgregarActividad(actividad4);
            //5
            DateTime fecha5 = new DateTime(2023, 06, 05);
            DateTime fechaConfirmacion5 = new DateTime(2023, 05, 25);
            Actividad actividad5 = new Actividad_Hostel("Padel", "Padel en Cancha del
Hostel", fecha5, 4, 15, "Actividad_Hostel",
                "Ricardo Dentone", "Cancha Cerrada del Hostel", false, 200);
            AgregarActividad(actividad5);
            //6
            DateTime fecha6 = new DateTime(2023, 06, 06);
            DateTime fechaConfirmacion6 = new DateTime(2023, 05, 25);
            Actividad actividad6 = new Actividad_Hostel("Merienda para dos", "Merienda
en cafeteria del Hostel", fecha6, 30, 1, "Actividad_Hostel",
                "Nelson Dentone", "Cafeteria", false, 500);
            AgregarActividad(actividad6);
            //7
            DateTime fecha7 = new DateTime(2023, 06, 07);
            DateTime fechaConfirmacion7 = new DateTime(2023, 05, 25);
            Actividad actividad7 = new Actividad_Hostel("Caminata", "Recorrida por la
```

```
ciudad", fecha7, 10, 15, "Actividad_Hostel",
                "Roberto Dentone", "Afueras del Hostel", true, 100);
            AgregarActividad(actividad7);
            //8
            DateTime fecha8 = new DateTime(2023, 06, 08);
            DateTime fechaConfirmacion8 = new DateTime(2023, 05, 25);
            Actividad actividad8 = new Actividad_Hostel("Museo", "Visita al museo de la
ciudad", fecha8, 20, 18, "Actividad_Hostel",
                "Juan Dentone", "Museo de la Ciudad", false, 1001);
            AgregarActividad(actividad8);
            //9
            DateTime fecha9 = new DateTime(2023, 06, 09);
            DateTime fechaConfirmacion9 = new DateTime(2023, 05, 25);
            Actividad actividad9 = new Actividad_Hostel("Masajes", "Masajes para
huespedes", fecha9, 4, 18, "Actividad_Hostel",
                "Osvaldo Dentone", "Spa Hostel", false, 1000);
            AgregarActividad(actividad9);
            //10
            DateTime fecha10 = new DateTime(2023, 06, 10);
            DateTime fechaConfirmacion10 = new DateTime(2023, 05, 25);
            Actividad actividad10 = new Actividad_Hostel("Yoga", "Yoga a la mañana",
fecha10, 10, 18, "Actividad_Hostel",
                "Christian Dentone", "jardin del Hostel", true, 150);
            AgregarActividad(actividad10);


            //ACTIVIDADES TERCERIZADAS
            //1
            DateTime fecha11 = new DateTime(2023, 06, 11);
            DateTime fechaConfirmacion11 = new DateTime(2023, 05, 20);
            Actividad actividad11 = new Actividad_Tercerizada("Paintball", "Juego de
equipo", fecha11, 11, 18,
                "Actividad_Tercerizada", _proveedores[0], true, fechaConfirmacion2,
500);
            AgregarActividad(actividad11);
            //2
            DateTime fecha12 = new DateTime(2023, 06, 12);
            DateTime fechaConfirmacion12 = new DateTime(2023, 05, 21);
            Actividad actividad12 = new Actividad_Tercerizada("Bungee Jumping",
"Extremo", fecha12, 11, 18,
                "Actividad_Tercerizada", _proveedores[0], true, fechaConfirmacion2,
200);
            AgregarActividad(actividad12);
            //3
            DateTime fecha13 = new DateTime(2023, 06, 13);
            DateTime fechaConfirmacion13 = new DateTime(2023, 05, 22);
            Actividad actividad13 = new Actividad_Tercerizada("Paracaidismo", "Exremo",
fecha13, 11, 18,
                "Actividad_Tercerizada", _proveedores[0], true, fechaConfirmacion13,
100);
            AgregarActividad(actividad13);
            //4
            DateTime fecha17 = new DateTime(2023, 06, 14);
            DateTime fechaConfirmacion17 = new DateTime(2023, 05, 25);
            Actividad actividad17 = new Actividad_Tercerizada("Museos de noche", "Paseo
```

```
por museos nocturnos", fecha17, 11, 18,
                "Actividad_Tercerizada", _proveedores[1], true, fechaConfirmacion17,
400);
            AgregarActividad(actividad17);
            //5
            DateTime fecha18 = new DateTime(2023, 06, 15);
            DateTime fechaConfirmacion18 = new DateTime(2023, 05, 25);
            Actividad actividad18 = new Actividad_Tercerizada("Lugares Historicos",
"Paseo por puntos emblematicos de la ciudad", fecha18, 11, 18,
                "Actividad_Tercerizada", _proveedores[1], true, fechaConfirmacion18,
500);
            AgregarActividad(actividad18);
            //6
            DateTime fecha19 = new DateTime(2023, 06, 16);
            DateTime fechaConfirmacion19 = new DateTime(2023, 05, 25);
            Actividad actividad19 = new Actividad_Tercerizada("Atardecer", "Puesta de
sol con musica y comida en la playa sur", fecha19, 11, 18,
                "Actividad_Tercerizada", _proveedores[1], true, fechaConfirmacion19,
600);
            AgregarActividad(actividad19);
            //7
            DateTime fecha14 = new DateTime(2023, 06, 17);
            DateTime fechaConfirmacion14 = new DateTime(2023, 05, 23);
            Actividad actividad14 = new Actividad_Tercerizada("Samba time", "Noche en la
roda de samba ", fecha14, 11, 18,
                "Actividad_Tercerizada", _proveedores[2], true, fechaConfirmacion14,
700);
            AgregarActividad(actividad14);
            //8
            DateTime fecha15 = new DateTime(2023, 06, 18);
            DateTime fechaConfirmacion15 = new DateTime(2023, 05, 24);
            Actividad actividad15 = new Actividad_Tercerizada("Pagodinho", "Pagode en
pedra do sal", fecha15, 11, 18,
                "Actividad_Tercerizada", _proveedores[2], true, fechaConfirmacion15,
800);
            AgregarActividad(actividad15);
            //9
            DateTime fecha16 = new DateTime(2023, 06, 19);
            DateTime fechaConfirmacion16 = new DateTime(2023, 05, 25);
            Actividad actividad16 = new Actividad_Tercerizada("Baile funk", "Baile en la
comunidad", fecha16, 11, 18,
                "Actividad_Tercerizada", _proveedores[2], true, fechaConfirmacion16,
900);
            AgregarActividad(actividad16);
            //10
            DateTime fecha20 = new DateTime(2023, 06, 20);
            DateTime fechaConfirmacion20 = new DateTime(2023, 05, 25);
            Actividad actividad20 = new Actividad_Tercerizada("Futbol", "Futbol 5",
fecha20, 11, 18,
                "Actividad_Tercerizada", _proveedores[3], true, fechaConfirmacion20,
600);
            AgregarActividad(actividad20);
            //11
            DateTime fecha21 = new DateTime(2023, 06, 21);
            DateTime fechaConfirmacion21 = new DateTime(2023, 05, 25);
```

```
            Actividad actividad21 = new Actividad_Tercerizada("Surf", "Clases de surf",
fecha21, 11, 18,
                "Actividad_Tercerizada", _proveedores[3], true, fechaConfirmacion21,
500);
            AgregarActividad(actividad21);
            //12
            DateTime fecha22 = new DateTime(2023, 06, 22);
            DateTime fechaConfirmacion22 = new DateTime(2023, 05, 25);
            Actividad actividad22 = new Actividad_Tercerizada("Futvoley", "Clases de
futvoley", fecha22, 11, 18,
                "Actividad_Tercerizada", _proveedores[3], true, fechaConfirmacion22,
700);
            AgregarActividad(actividad22);
            //13
            DateTime fecha23 = new DateTime(2023, 06, 23);
            DateTime fechaConfirmacion23 = new DateTime(2023, 05, 25);
            Actividad actividad23 = new Actividad_Tercerizada("Trekking", "Senderismo
por el cerro de la buena vista", fecha23, 11, 18,
                "Actividad_Tercerizada", _proveedores[4], true, fechaConfirmacion23,
500);
            AgregarActividad(actividad23);
            //14
            DateTime fecha24 = new DateTime(2023, 06, 24);
            DateTime fechaConfirmacion24 = new DateTime(2023, 05, 25);
            Actividad actividad24 = new Actividad_Tercerizada("Escalada", "Escalada al
Pan de Azucar", fecha24, 11, 18,
                "Actividad_Tercerizada", _proveedores[4], true, fechaConfirmacion24,
700);
            AgregarActividad(actividad24);
            //15
            DateTime fecha25 = new DateTime(2023, 06, 25);
            DateTime fechaConfirmacion25 = new DateTime(2023, 05, 25);
            Actividad actividad25 = new Actividad_Tercerizada("Buceo", "Buceo en la
playa sur", fecha25, 11, 18,
                "Actividad_Tercerizada", _proveedores[4], true, fechaConfirmacion25);
            AgregarActividad(actividad25);
        }

        //Metodo para agregar Actividad, validando que sea única y asegurándose que no
sea nula.
        private void AgregarActividad(Actividad actividad)
        {
            if (actividad == null)
            {
                throw new Exception("La actividad recibida no es vàlida.");
            }
            if (_actividades.Contains(actividad)) // Validación unicidad ID Actividad
utilizando el Equals definido en Actividad
            {
                throw new Exception($"La actividad con id: {actividad._id} ya esta
registrada.");
            }
            actividad.Validar();
            _actividades.Add(actividad);
        }
```

```
// Metodo para precargar Agenda
private void PrecargarAgenda()
{
    Usuario usuario = _usuarios[0];
    if (usuario is Huesped)
    {
        Huesped huesped = (Huesped)usuario;
        Agenda agenda1 = new Agenda(huesped, _actividades[0], "confirmada");
        AgregarAgenda(agenda1);
    }

    Usuario usuario2 = _usuarios[1];
    if (usuario2 is Huesped)
    {
        Huesped huesped2 = (Huesped)usuario2;
        Agenda agenda2 = new Agenda(huesped2, _actividades[1], "pendiente");
        AgregarAgenda(agenda2);
    }
}

//Método para agregar Agenda, previa validación
private void AgregarAgenda(Agenda agenda)
{
    if (agenda == null)
    {
        throw new Exception("La informacion recibida no es vàlida.");
    }
    if (_agenda.Contains(agenda))
    {
        throw new Exception($"La actividad ya esta agendada");
    }
    agenda.Validar();
    _agenda.Add(agenda);
}

// Método para obtener huesped por cédula
public Huesped? ObtenerHuespedPorCedula(string cedula)
{
    foreach (Usuario item in _usuarios)
    {
        if (item is Huesped)
        {
            Huesped huesped = (Huesped)item;

            if (huesped.NumeroDocumento == cedula)
            {
                return huesped;
            }
        }
    }
    return null;
}

//Método para obener Actividad por ID
```

```
public Actividad? ObtenerActividadPorId(int id)
{
    foreach (Actividad item in _actividades)
    {
        if (item._id == id)
        {
            return item;
        }
    }
    return null;
}

// Método para usar en segunda parte
public decimal CalcularCostoActividad(string numeroDocumento, int id)
{

    Actividad actividad = ObtenerActividadPorId(id);

    decimal costoFinalActividad = actividad.Costo;

    Huesped huesped = ObtenerHuespedPorCedula(numeroDocumento);

    int fidelizacion = int.Parse(huesped.Fidelizacion);

    switch (fidelizacion)
    {
        case 2:
            costoFinalActividad = Math.Round(actividad.Costo * 0, 9);
            break;
        case 3:
            costoFinalActividad = Math.Round(actividad.Costo * 0, 85);
            break;
        case 4:
            costoFinalActividad = Math.Round(actividad.Costo * 0, 8);
            break;
    }
    return costoFinalActividad;
}

//Metodo para listar y ordenar Proveedores utilizando el Compare de Proveedor
public IEnumerable<Proveedor> ProveedoresOrdenados()
{
    List<Proveedor> aux = new List<Proveedor>();
    foreach (Proveedor item in _proveedores)
    {
        aux.Add(item);
    }
    aux.Sort();
    return aux;
}

//Metodo para obtener Proveedor por ID
public Proveedor? ObtenerProveedorPorId(int id)
{
    foreach (Proveedor item in _proveedores)
```

```
            {
                if (item._id == id)
                {
                    return item;
                }
            }
            return null;
        }

        //Metodo para listar y ordenar Actividades por fecha y costo.
        public IEnumerable<Actividad> ListarActividadesFechaCosto(DateTime fecha1,
DateTime fecha2, decimal costoPedido)
        {
            List<Actividad> aux = new List<Actividad>();
            foreach (Actividad item in _actividades)
            {

                if (item.FechaActividad >= fecha1 && item.FechaActividad <= fecha2 &&
item.Costo > costoPedido)
                {
                    aux.Add(item);
                }
            }
            aux.Sort();
            return aux;
        }

        //Para generar una lista auxiliar solo con Huéspedes.
        public IEnumerable<Huesped> ListaHuespedes()
        {
            List<Huesped> aux = new List<Huesped>();
            foreach (Usuario item in _usuarios)
            {
                if (item is Huesped)
                {
                    Huesped huesped = (Huesped)item;
                    aux.Add(huesped);
                }
            }
            return aux;
        }
    }
}
```

USUARIO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Dominio
{
    //Creación de la clase Usuario, con la interfaz IValidable para garantizar la
```

existencia del método Validar(), y la interfaz IEquatable para que Conntains verifique por el Email

```
    public abstract class Usuario: IValidable, IEquatable<Usuario>
    {
        public string Email { get; set; }
        public string Contrasenia { get; set; }

        //Constructor
        public Usuario(string email, string contrasenia)
        {
            Email = email;
            Contrasenia = contrasenia;
        }

        //Método validar
        public void Validar()
        {
            validarEmail();
            validarContrasenia();
        }

        //Método para validar Email según los requerimientos
        public void validarEmail()
        {

            if (Email.Contains("@") && Email.IndexOf("@") > 0 && Email.LastIndexOf("@")
< Email.Length - 1)
            {

            }
            else
            {
                throw new Exception("La dirección de correo electrónico no es válida.");
            }

        }

        // Método para validar Contraseña según los requerimientos
        public void validarContrasenia()
        {
            if ( Contrasenia.Length < 8)
            {
                throw new Exception("La Contraseña debe contener al menos 8
caracteres");
            }

        }

        // Definición del Contains por Email
        public bool Equals(Usuario? other)
        {
            return other != null && Email == other.Email;
        }}}
```