Obligatorio Programación 2

Bruno Huber Garín
Nro. Estudiante: 301238 Grupo: N2D
REM
Docente: Arturo Velarde

URL somee : http://bh301238Obligatorio2.somee.com
http://www.bh301238Obligatorio2.somee.com

Diagrama de Clases

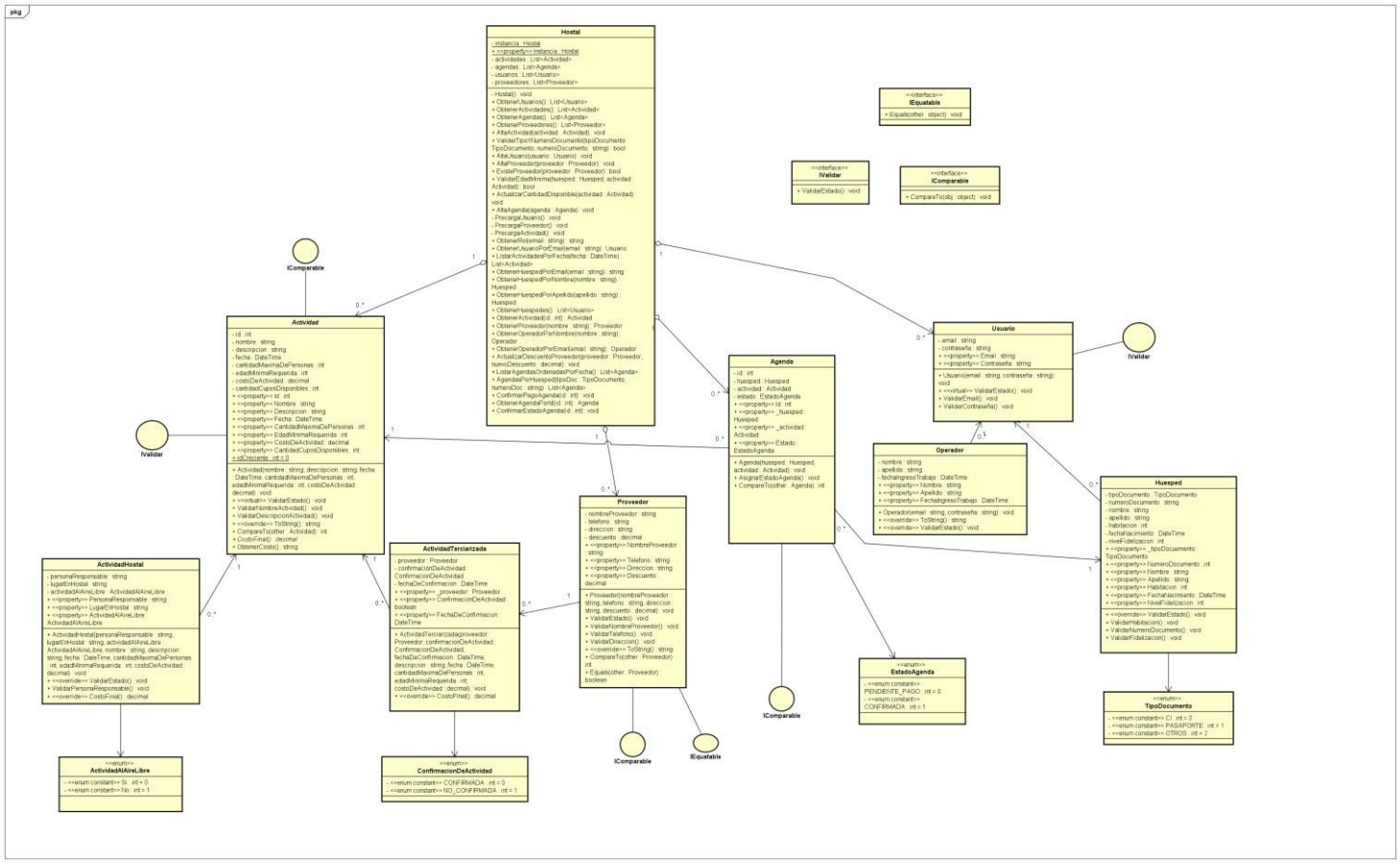
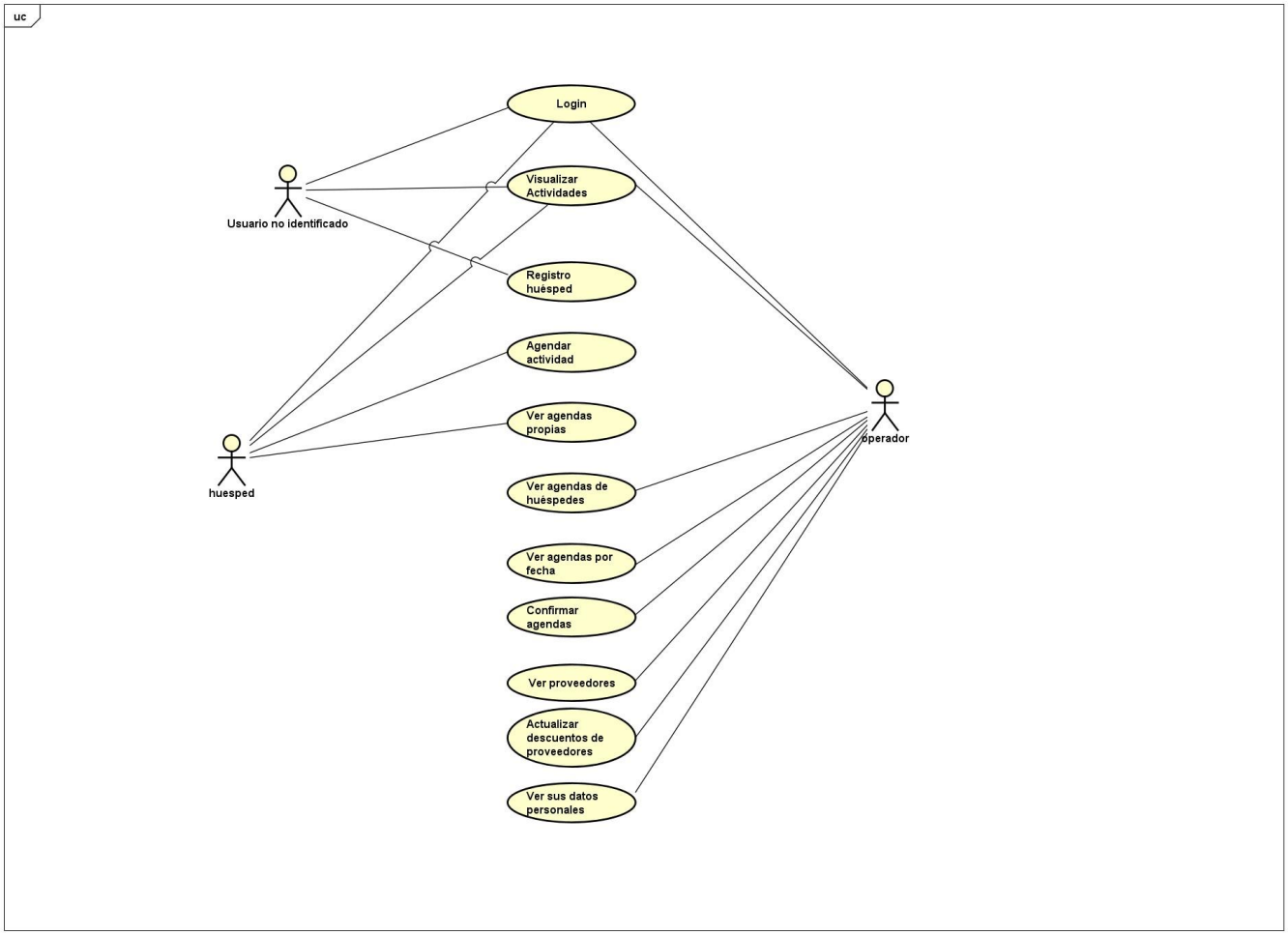


Diagrama casos de uso:



Datos precargados

Precarga de Usuario(Huésped):

```
 AltaUsuario(new Huesped("jorgeRodriguez@gmail.com", "jorgeRodriguez123",
TipoDocumento.CI, "43681573", "Jorge", "Rodríguez", 102, new DateTime(1990, 06, 04),
2));
```

```
        AltaUsuario(new Huesped("martin_delcampo@hotmail.com", "del_campo555",
TipoDocumento.PASAPORTE, "52613194", "Martin", "Del Campo", 25, new DateTime(1998,
02, 27), 2));
```

Precarga de Usuario (Operador):

```
AltaUsuario(new Operador("ricardo_operador@gmail.com", "ricardo12345", "Ricardo",
"López", new DateTime(2006, 03, 13)));
        AltaUsuario(new Operador("federico_operador@gmail.com", "fede12345",
"Federico", "Roque", new DateTime(2016, 10, 22)));
```

Precarga Actividad(De Hostal):

```
AltaActividad(new ActividadHostal("Fútbol", "Partido entre huespedes",
DateTime.Today, 22, 18, 250, "Alberto Martínez", "Cancha Fútbol",
ActividadAlAireLibre.Si));

 AltaActividad(new ActividadHostal("Yoga", "Ejercicio relajación", new DateTime(2024,
07, 23), 16, 21, 300, "Mónica Dominguez", "Sala 3", ActividadAlAireLibre.No));

AltaActividad(new ActividadHostal("Baile", "Clases de baile", new DateTime(2023, 11,
06), 10, 16, 450, "Manuel Ortega", "Salón de baile", ActividadAlAireLibre.No));
AltaActividad(new ActividadHostal("kayak", "Paseo en Kayak", new DateTime(2023, 06,
15), 10, 18, 500, "Daniela Molina", "Río Conde", ActividadAlAireLibre.Si));
AltaActividad(new ActividadHostal("Tennis", "Clases particulares de tennis",
DateTime.Today, 4, 12, 680, "Mariano Pernía", "Canchas de Tennis",
ActividadAlAireLibre.Si));

AltaActividad(new ActividadHostal("Natación", "Espacio de recreación en piscina", new
DateTime(2024, 03, 30), 15, 12, 850, "Victor Hugo Morales", "Piscinas del hostal",
ActividadAlAireLibre.No));

AltaActividad(new ActividadHostal("Spinning", "Sesión de spinning con profesor", new
DateTime(2022, 09, 14), 9, 16, 200, "Alfredo Montes de Oca", "Gimnasio del hostal",
ActividadAlAireLibre.No));

AltaActividad(new ActividadHostal("Clases de Zumba", "Sesión de zumba con profesora",
new DateTime(2024, 04, 30), 13, 18, 350, "Rosa Luna", "Gimnasio del hostal",
ActividadAlAireLibre.No));

AltaActividad(new ActividadHostal("Cine", "Noche de cine", new DateTime(2023, 12,
20), 25, 5, 500, "Carlos Reyes", "Salón de espectaculos", ActividadAlAireLibre.No));
AltaActividad(new ActividadHostal("Paseo a Caballo", "Paseo por las sierras a
caballo", new DateTime(2023, 08, 12), 6, 16, 135, "Marcos Pérez", "Predio del
hostal", ActividadAlAireLibre.Si));

AltaActividad(new ActividadHostal("Paseuuo a Caballo", "Paseo por las sierras a
caballo", new DateTime(2023, 08, 12), 6, 16, 0, "Marcos Pérez", "Predio del hostal",
ActividadAlAireLibre.Si));
```

Precarga Actividad(Terciarizadas):

```
 AltaActividad(new ActividadTerciarizada("Alpinismo", "Escalar las sierras", new
DateTime(2022, 10, 12), 5, 18, 2135, proveedores[0],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 10, 10)));

AltaActividad(new ActividadTerciarizada("Paracaidismo", "Tirarse en paracaidas", new
```

```
DateTime(2022, 06, 22), 3, 21, 1950, proveedores[0],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 06, 10)));

AltaActividad(new ActividadTerciarizada("Pesca en bote", "Salida de pesca por el
río", new DateTime(2023, 11, 02), 10, 15, 980, proveedores[0],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 10, 29)));

AltaActividad(new ActividadTerciarizada("Paintball", "Contienda de Paintball", new
DateTime(2023, 07, 15), 20, 18, 3200, proveedores[1],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 07, 10)));

AltaActividad(new ActividadTerciarizada("Paseo por el bosque", "Un paseo por el
interior del bosque", new DateTime(2023, 12, 03), 25, 8, 650, proveedores[1],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 11, 28)));

AltaActividad(new ActividadTerciarizada("Concierto de rock", "Salida a evento de
rock", new DateTime(2023, 09, 15), 30, 18, 2200, proveedores[1],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 08, 10)));

AltaActividad(new ActividadTerciarizada("Parque de diversiones", "Salida a parque de
diversiones", new DateTime(2023, 10, 10), 20, 5, 1000, proveedores[2],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 08, 15)));

AltaActividad(new ActividadTerciarizada("Carrera en moto", "Evento de carreras en
moto", new DateTime(2023, 11, 14), 12, 18, 3000, proveedores[2],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 11, 03)));

AltaActividad(new ActividadTerciarizada("Parque acuático", "Salida al parque
acuático", new DateTime(2024, 01, 12), 30, 10, 4500, proveedores[2],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 12, 26)));

AltaActividad(new ActividadTerciarizada("Visita a Iglesia Matriz", "Salida a la
Iglesia Matriz", new DateTime(2023, 10, 04), 25, 5, 1500, proveedores[3],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 09, 26)));

AltaActividad(new ActividadTerciarizada("Salida a la playa", "Una salida en familia a
la playa", new DateTime(2024, 02, 05), 35, 5, 1200, proveedores[3],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 12, 22)));

AltaActividad(new ActividadTerciarizada("Casamiento Falso", "Evento en casamiento
simulado", new DateTime(2023, 07, 07), 40, 18, 3500, proveedores[3],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 04, 30)));

AltaActividad(new ActividadTerciarizada("Paseo en helicóptero", "Recorrido en
helicóptero por la ciudad", new DateTime(2023, 08, 09), 4, 18, 6500, proveedores[4],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 07, 11)));
AltaActividad(new ActividadTerciarizada("Teatro", "Salida a evento
de teatro", new
DateTime(2023, 11, 04), 15, 18, 2650, proveedores[4],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 09, 01)));

AltaActividad(new ActividadTerciarizada("Salida al Estadio", "Salida a partido de
fútbol profesional", new DateTime(2023, 10, 22), 15, 12, 5300, proveedores[4],
ConfirmacionDeActividad.CONFIRMADA, new DateTime(2023, 09, 13)));
AltaActividad(new ActividadTerciarizada("Salida al Circo", "Salida a circo de
barrio", new DateTime(2023, 10, 12), 15, 12, 5300, proveedores[4],
ConfirmacionDeActividad.NO_COFIRMADA, null));
```

Precarga de Proveedores:

```
AltaProveedor(new Proveedor("DreamWorks S.R.L.", "23048549", "Suarez 3380 Apto 304",
```

```
10));

 AltaProveedor(new Proveedor("Estela Umpierrez S.A.", "33459678", "Lima 2456", 7));

AltaProveedor(new Proveedor("TravelFun", "29152020", "Misiones 1140", 9));

AltaProveedor(new Proveedor("Rekreation S.A.", "29162019", "Bacacay 1211", 11));
AltaProveedor(new Proveedor("Alonso & Umpierrez", "24051920", "18 de Julio 1956 Apto
4", 10));

AltaProveedor(new Proveedor("Electric Blue", "26018945", "Cooper 678", 5));
AltaProveedor(new Proveedor("Lúdica S.A.", "26142967", "Dublin 560", 4));
AltaProveedor(new Proveedor("Gimenez S.R.L.", "29001010", "Andes 1190", 7));

AltaProveedor(new Proveedor("papaNoel", "22041120", "Agraciada 2512 Apto. 1", 8));
AltaProveedor(new Proveedor("Norberto Molina", "22001189", "Paraguay 2100", 9));
```

Código fuente:

// Hostal.css : ----------

```
private static Hostal instancia;

    // Aplico patrón singleton
public static Hostal Instancia
    {
get
{
            if (instancia == null)
            {
                instancia = new Hostal();
instancia.PrecargaUsuario();
instancia.PrecargaProveedor();
instancia.PrecargaActividad();
                //instancia.PrecargaAgenda();


            }
            return instancia;
        }
    }


    //inicio las listas

    private List<Actividad> actividades;
private List<Agenda> agendas;     private
List<Usuario> usuarios;     private
List<Proveedor> proveedores;

    private Hostal()
    {
        actividades = new List<Actividad>();
agendas = new List<Agenda>();         usuarios
= new List<Usuario>();         proveedores =
new List<Proveedor>();
    }

    // ---------------- Metodos para acceder a las listas --------------
public List<Usuario> ObtenerUsuarios() { return usuarios; }     public
List<Actividad> ObtenerActividades() { return actividades; }

    public List<Agenda> ObtenerAgendas() { return agendas; }

    public List<Proveedor> ObtenerProveedores() { return proveedores; }


    // --- Metodo para dar de alta actividades ---------------------
public void AltaActividad(Actividad actividad)
    {
        if (actividad != null) // Verifico que la actividad que se quiere
ingresar no sea nula
        {
            actividad.CostoFinal();
actividad.ObtenerCosto();
            actividad.ValidarEstado(); //Valido si se cumple con los
requisitos antes de agregar una Actividad
actividades.Add(actividad);             actividad.ToString();
```

```
         }
else
{
            throw new Exception("No existe la actividad");
        }
}
    public bool ValidarTipoYNumeroDocumento(TipoDocumento tipoDocumento,
string numeroDocumento) //metodo para validar que la combinación entre el
numero de documento y el tipo de documento del Huesped sea unica en el sistema
    {
        ObtenerUsuarios();
        foreach (Usuario usuario in usuarios) // recorro la lista de
usuarios         {
            if (usuario is Huesped huesped)// Corroboro que el usuario que
estoy buscado sea de tipo Huesped
            {
                if (huesped._tipoDocumento == tipoDocumento &&
huesped.NumeroDocumento == numeroDocumento)
                {
                    return false; // la combinaciòn ya existe en la lista de
usuarios                 }
            }
}
        return true; // se puede agregar el usuario
    }

// --- Metodo para dar de alta Usuarios ---------------------

    public void AltaUsuario(Usuario usuario)
    {
        ObtenerUsuarios();
if (usuario != null)
        {
            if (usuario is Huesped huesped)
            {
                if (ValidarTipoYNumeroDocumento(huesped._tipoDocumento,
huesped.NumeroDocumento))// Si la validacion devuelve true, se sigue
adelante                 {
                    huesped.ValidarEstado(); //Valido si se cumple con los
requisitos antes de agregar un Usuario
usuarios.Add(usuario);                      usuario.ToString();
                }
            }

            else if (usuario is Operador operador)
            {
                operador.ValidarEstado();
usuarios.Add(usuario);
            }

            else
{
                throw new Exception("Usuario ya registrado");
            }
         }
else
{
            throw new Exception("El usuario no existe");
        }
```

```
    }

    // --- Metodo para dar de alta Proveedores --------------------

    public void AltaProveedor(Proveedor proveedor)
    {
        if (ExisteProveedor(proveedor)) // verifico que el proveedor no exista
en el sistema antes de agregarlo
        {
            throw new Exception("El proveedor ya existe en la lista");
        }
else         {
            proveedor.ValidarEstado();
proveedores.Add(proveedor);             proveedor.ToString();

        }

    }

    // ---- Metodo para corroborar que no se repite el nombre del proveedor
------------------
    public bool ExisteProveedor(Proveedor proveedor)
    {
        bool existeProveedor = false;

        foreach (var p in proveedores)
        {
            if (p.NombreProveedor == proveedor.NombreProveedor)
            {
                existeProveedor = true;
break;
            }
}
        return existeProveedor;
    }




    /// // -------------- Para dar de alta una agenda ----------------------
-----------------------------------------------

    public bool ValidarEdadMinima(Huesped huesped, Actividad actividad)
{
        bool esApto = true;
        DateTime fechaActual = DateTime.Now;

        int edad = fechaActual.Year - huesped.FechaNacimiento.Year;

        if (fechaActual.Month < huesped.FechaNacimiento.Month ||
(fechaActual.Month == huesped.FechaNacimiento.Month && fechaActual.Day <
huesped.FechaNacimiento.Day))
        {
            edad--; // Ajuste si aún no ha cumplido años en el año actual
}
        if (actividad.EdadMinimaRequerida > edad)
        {
            esApto = false;
```

```
        }
        return esApto;
    }


    public void ActualizarCantidadDisponible(Actividad actividad)
    {
        foreach (Actividad item in actividades)
        {
            if (item.Id == actividad.Id)
            {
                item.CantidadCuposDisponibles--;
            }
        }
    }
    public void AltaAgenda(Agenda agenda)
    {
        if (agenda != null)
        {
            ValidarEdadMinima(agenda._huesped, agenda._actividad);
            if (agenda._actividad.CantidadCuposDisponibles > 0) {

                    agendas.Add(agenda);
                    ActualizarCantidadDisponible(agenda._actividad);
agenda.AsignarEstadoAgenda();
            }
        }
else         {
            throw new Exception("La agenda no existe");
        }
}
public string ObtenerRol(string email)
    {
        var usuario = ObtenerUsuarioPorEmail(email);

            if (usuario is Huesped)
            {
                return "huesped";
            }
            else if (usuario is Operador)
            {
                return "operador";
            }

        return "ninguno";
    }
    public Usuario ObtenerUsuarioPorEmail(string email)
    {
        foreach (Usuario usuario in usuarios)
        {
            if (usuario.Email == email)
            {
                return usuario;

            }

}
        return null;
```

```
    }



    public List<Actividad> ListarActividadesPorFecha(DateTime fecha)
    {
        List<Actividad> listaActividades = new List<Actividad>();
foreach (var actividad in listaActividades)
        {
            if (actividad.Fecha == fecha)
            {
                listaActividades.Add(actividad);
            }
}
        return listaActividades;
    }
    public Huesped ObtenerHuespedPorEmail(string email)
    {
        Huesped huesped = null;

        foreach (Usuario usuario in usuarios)
        {
            if (usuario is Huesped huespedActual && huespedActual.Email ==
email)             {
                huesped = huespedActual;
break;
            }
        }
        return huesped;
    }


    public Huesped ObtenerHuespedPorNombre(string nombre)
    {
        Huesped huesped = null;

        foreach (Usuario usuario in usuarios)
{
            if (usuario is Huesped huespedActual && huespedActual.Nombre ==
nombre)              {
                huesped = huespedActual;
break;
            }
        }
        return huesped;
    }
    public Huesped ObtenerHuespedPorApellido(string apellido)
    {
        Huesped huesped = null;

        foreach (Usuario usuario in usuarios)
        {
            if (usuario is Huesped huespedActual && huespedActual.Apellido
== apellido)
{
                huesped = huespedActual;
break;
```

```
            }
        }
        return huesped;
    }
    public List<Usuario> ObtenerHuespedes()
    {
        List<Usuario> huespedes = new List<Usuario>();

        foreach (Usuario usuario in usuarios)
        {
            if (usuario is Huesped huesped)
            {
                huespedes.Add(huesped);
            }
        }
        return huespedes;
    }
    public Actividad ObtenerActividad(int id)
    {
        Actividad actividad = null;
        foreach (Actividad item in actividades)
        {
            if (item.Id == id)
            {
                return item;

            }
}
        return actividad;
    }


    public Proveedor ObtenerProveedor(string nombre)
    {
        foreach (Proveedor item in proveedores)
{
            if (item.NombreProveedor == nombre)
            {
                return item;

            }
}
        return null;
    }
    public Operador ObtenerOperadorPorNombre(string nombre)
    {
        Operador operador = null;

        foreach (Usuario usuario in usuarios)
        {
            if (usuario is Operador operadorActual && operadorActual.Nombre
== nombre)
{
                operador = operadorActual;
break;
            }
        }
        return operador;
    }
```

```
    public Operador ObtenerOperadorPorEmail(string email)
    {
        Operador operador = null;

        foreach (Usuario usuario in usuarios)
        {
            if (usuario is Operador operadorActual && operadorActual.Email
== email)
{
                operador = operadorActual;
break;
            }
        }
        return operador;
    }


    public void ActualizarDescuentoProveedor(Proveedor proveedor, decimal
nuevoDescuento)     {
        var proveedorEntidad = proveedores.Find(x => x.NombreProveedor ==
proveedor.NombreProveedor);         if (proveedorEntidad is null)
        {
            throw new Exception("Proveedor no encontrado");
        }

        proveedorEntidad.Descuento = nuevoDescuento;
    }
    public List<Agenda> ListarAgendasOrdenadasPorFecha()
    {
        ObtenerAgendas();

        agendas.Sort();

        return agendas;
    }
    public List<Agenda> AgendasPorHuesped(TipoDocumento tipoDoc, string
numeroDoc)
    {
        List<Agenda> listaAgendas = new List<Agenda>();

        foreach (var agenda in agendas)
        {
            if (agenda != null && agenda._huesped != null &&
agenda._huesped._tipoDocumento == tipoDoc && agenda._huesped.NumeroDocumento
== numeroDoc)
{
                listaAgendas.Add(agenda);
            }
}
        return listaAgendas;
    }
    public void ConfirmarPagoAgenda(int id)
    {
        foreach (var agenda in agendas)
        {
            if (agenda.Id == id && agenda.Estado ==
EstadoAgenda.PENDIENTE_PAGO)
            {
                agenda.Estado = EstadoAgenda.CONFIRMADA;
```

```
            }

        }

    }
    public Agenda ObtenerAgendaPorId(int id)
    {
        foreach (var agenda in agendas)
        {
            if (agenda.Id == id)
            {
                return agenda;
            }
}
        return null; // Si no se encuentra la agenda, se devuelve null
    }
    public void ConfirmarEstadoAgenda(int id)
    {
        var agenda = ObtenerAgendaPorId(id);

        if (agenda == null)
        {
            throw new Exception("No se encontró la agenda.");
        }
        if (agenda.Estado != EstadoAgenda.PENDIENTE_PAGO)
        {
            throw new Exception("La agenda no se puede confirmar porque no
está en estado pendiente de pago.");
        }
        agenda.Estado = EstadoAgenda.CONFIRMADA;


    }
}
```

// Actividad.cs : ----------

```
public abstract class Actividad : IValidar, IComparable<Actividad>
{
    private int id; public int Id { get => id;}

    private string nombre; public string Nombre { get => nombre; }

    private string descripcion; public string Descripcion { get =>
descripcion; }

    private DateTime fecha; public DateTime Fecha { get => fecha; }
private int cantidadMaximaDePersonas; public int
CantidadMaximaDePersonas { get => cantidadMaximaDePersonas; set =>
cantidadMaximaDePersonas = value; }

    private int edadMinimaRequerida; public int EdadMinimaRequerida { get =>
edadMinimaRequerida; }

    private decimal costoDeActividad; public decimal CostoDeActividad { get
=> costoDeActividad; }
```

```
    private int cantidadCuposDisponibles; public int
CantidadCuposDisponibles { get => cantidadCuposDisponibles; set =>
cantidadCuposDisponibles = value; }


    private static int idCreciente = 0; // genero una variable auxiliar para
autogenerar id creciente en cada instancia de Actividad.



    public Actividad(string nombre, string descripcion, DateTime fecha, int
cantidadMaximaDePersonas, int edadMinimaRequerida, decimal costoDeActividad)
    {

        this.id = idCreciente++;
this.nombre = nombre;         this.descripcion
= descripcion;         this.fecha = fecha;
        this.cantidadMaximaDePersonas = cantidadMaximaDePersonas;
this.cantidadCuposDisponibles = cantidadMaximaDePersonas;
this.edadMinimaRequerida = edadMinimaRequerida;         this.costoDeActividad
= costoDeActividad;

    }
    public virtual void ValidarEstado() {
        ValidarNombreActividad();
        ValidarDescripcionActividad();
        //ValidarFechaActividad();
    }

    public void ValidarNombreActividad()

{
        if (string.IsNullOrEmpty(Nombre)) {
            throw new Exception("El campo Nombre no puede ser vacío");

        }
        if (Nombre.Length > 25)
        {
            Console.WriteLine("El Nombre no puede contener mas de 25
caracteres");         }

    }
    public void ValidarDescripcionActividad()
    {
        if (string.IsNullOrEmpty(Descripcion))
        {
            throw new Exception("El campo Descripción no puede ser vacío");
        }
    }

    public override string ToString()
    {
        return
                $"Id: {id}\n" +
                $"Nombre: {nombre}\n" +
                $"Descripción: {descripcion}\n" +
                $"Fecha: {fecha.ToString("d")}\n" +
                $"Cantidad máxima de personas: {CantidadMaximaDePersonas}\n"
+
```

```
                $"Edad minima: {edadMinimaRequerida}\n"+
                $"--------------------------------\n";
                //$"{costoDeActividad}\n" +
    }
    public int CompareTo(Actividad? other)
    {
        return CostoDeActividad.CompareTo(other.CostoDeActividad) * -1;
    }
    public abstract decimal CostoFinal();

    public string ObtenerCosto()
    {
        if (CostoDeActividad > 0)
        {
            return CostoDeActividad.ToString();
        }
else         {
            return "actividad gratuita";
        }
    }



}
```

// ActividadAlAireLibre.cs : ----------

```
  namespace
ClasesObligatorio
{
    public enum ActividadAlAireLibre
    {
        Si,
        No
    }
}
```

// ActividadHostal.cs :

```
public class ActividadHostal : Actividad, IValidar
{
    private string personaResponsable; public string PersonaResponsable { get
=> personaResponsable; }

    private string lugarEnHostal; public string LugarEnHostal { get =>
lugarEnHostal; }
    private ActividadAlAireLibre actividadAlAireLibre; public
ActividadAlAireLibre ActividadAlAireLibre { get => actividadAlAireLibre; }


    public ActividadHostal( string nombre, string descripcion, DateTime
fecha, int cantidadMaximaDePersonas, int edadMinimaRequerida, decimal
costoDeActividad, string personaResponsable, string lugarEnHostal,
ActividadAlAireLibre actividadAlAireLibre) :base( nombre, descripcion, fecha,
cantidadMaximaDePersonas, edadMinimaRequerida, costoDeActividad)
    {
```

```
        this.personaResponsable = personaResponsable;
this.lugarEnHostal = lugarEnHostal;
        this.actividadAlAireLibre = actividadAlAireLibre;

    }


    public override void ValidarEstado()
    {
        base.ValidarEstado();
ValidarPersonaResponsable();
    }
    public void ValidarPersonaResponsable()

{
        if (string.IsNullOrEmpty(PersonaResponsable))
        {
            throw new Exception("Se debe indicar la persona responsable de
la actividad");

        }
    }

    public override decimal CostoFinal()
    {
            int nivelFidelizacion = 0;
decimal costoFinal= CostoDeActividad;
decimal descuento = 0;

                switch (nivelFidelizacion)
                {
case 1:
                        // sin descuento                            break;
case 2:                         descuento = costoFinal * 0.10m; // 10% de
descuento                         costoFinal -= descuento;
break;                      case 3:                          descuento =
costoFinal * 0.15m; // 15% de descuento                          costoFinal
-= descuento;                           break;                     case 4:
descuento = costoFinal * 0.20m; // 20% de descuento
costoFinal -= descuento;                          break;


}
            return costoFinal;


    }
```

// ActividadTerciarizada.cs : ----------

```
public class ActividadTerciarizada : Actividad
{
      private Proveedor proveedor; public Proveedor _proveedor{ get
=>proveedor; } // Utilizo "_" en vez de mayùscula para diferenciarlo de la
clase "Proveedor"
```

```
      private ConfirmacionDeActividad confirmacionDeActividad; public
ConfirmacionDeActividad ConfirmacionDeActividad { get =>
confirmacionDeActividad; set => confirmacionDeActividad = value; }
     private DateTime? fechaDeConfirmacion; public
DateTime?
FechaDeConfirmacion { get =>fechaDeConfirmacion; }

    public ActividadTerciarizada( string nombre, string descripcion, DateTime
fecha, int cantidadMaximaDePersonas, int edadMinimaRequerida, decimal
costoDeActividad, Proveedor proveedor, ConfirmacionDeActividad
confirmacionDeActividad, DateTime? fechaDeConfirmacion) : base( nombre,
descripcion, fecha, cantidadMaximaDePersonas, edadMinimaRequerida,
costoDeActividad)
    {
        this.proveedor = proveedor;
        this.confirmacionDeActividad = confirmacionDeActividad;

        if (confirmacionDeActividad == ConfirmacionDeActividad.CONFIRMADA)
{
            this.fechaDeConfirmacion = fechaDeConfirmacion;
        }
else          {
            this.fechaDeConfirmacion = null;
        }

    }
    public override decimal CostoFinal()
    {
        decimal costoFinal = CostoDeActividad;
        if (ConfirmacionDeActividad == 0 && _proveedor != null)
        {
            decimal descuento = proveedor.Descuento;
costoFinal *= descuento;
        }
        return costoFinal;
    }




}
```

// Agenda.cs : ----------

```
    public class Agenda : IComparable<Agenda>
    {
        private int id; public int Id { get => id; set => id = value; }
private Huesped huesped; public Huesped _huesped { get => huesped; set =>
huesped = value; }

        private Actividad actividad; public Actividad _actividad { get =>
actividad; set => actividad = value; }
```

```
        private EstadoAgenda estado; public EstadoAgenda Estado { get =>
estado; set => estado = value;}
        private Proveedor proveedor; public Proveedor Proveedor { get =>
proveedor; set => proveedor = value; }

        private static int idCreciente = 0;


        public Agenda(Huesped huesped, Actividad actividad)
        {
            this.id = idCreciente++;
this.huesped = huesped;
            this.actividad = actividad;
            AsignarEstadoAgenda();

        }



        public void AsignarEstadoAgenda()
        {
            if (actividad.CostoDeActividad > 0)
            {
                Estado = EstadoAgenda.PENDIENTE_PAGO;
            }
else
            {
                Estado = EstadoAgenda.CONFIRMADA;
            }
        }
        public int CompareTo(Agenda? other)
        {
            return actividad.Fecha.CompareTo(other.actividad.Fecha);
        }

    }
```

// ConfirmacionDeActividad.cs : ----------

```
     public enum
ConfirmacionDeActividad
    {
        CONFIRMADA,
        NO_COFIRMADA
    }
}
```

// EstadoAgenda.cs : ----------

```
  public enum
EstadoAgenda
    {
        PENDIENTE_PAGO,
        CONFIRMADA
    }
```

```
// Huesped.cs : ------------------------------------------------------
--
public class Huesped : Usuario, IValidar
{
    private TipoDocumento tipoDocumento; public TipoDocumento _tipoDocumento
{get => tipoDocumento; } // lo pongo con "_" para diferenciarlo del tipo de
dato
 private string numeroDocumento; public string NumeroDocumento { get =>
numeroDocumento; }

    private string nombre; public string Nombre {get => nombre ;}

       private string apellido; public string Apellido {get => apellido; }

       private int habitacion; public int Habitacion{get => habitacion; }
private DateTime fechaNacimiento; public DateTime FechaNacimiento { get
=> fechaNacimiento; }

       private int nivelFidelizacion; public int NivelFidelizacion { get =>
nivelFidelizacion; }

    public   Huesped(string   email,   string   contraseña,   TipoDocumento
tipoDocumento, string numeroDocumento, string nombre, string apellido, int
habitacion, DateTime fechaNacimiento, int nivelFidelizacion) :base( email,
contraseña)     {
         this.tipoDocumento= tipoDocumento;
this.numeroDocumento=numeroDocumento;
this.nombre=nombre;            this.apellido=apellido;
this.habitacion=habitacion;
this.fechaNacimiento = fechaNacimiento;
this.nivelFidelizacion = nivelFidelizacion;
    }




    public override void ValidarEstado()
    {
        base.ValidarEstado();
        ValidarHabitacion();
        ValidarNumeroDocumento(numeroDocumento);
        ValidarFidelizacion();
    }
    public void ValidarHabitacion()
    {
        if (Habitacion < 0)
        {
            throw new Exception("Se debe ingresar el número de habitación y
éste debe ser mayor que 0");
        }
    }
    public void ValidarNumeroDocumento(string numeroDocumento)
    {
        if (tipoDocumento == TipoDocumento.CI)
        {
            string valor = "1234567890";
            for (int i = 0; i < numeroDocumento.Length; i++)
```

```
            {
                if (!valor.Contains(numeroDocumento[i].ToString()))
                {
                    throw new Exception("El número de documento debe
contener caracteres numérico del 0 al 9");
                }
}
            if (NumeroDocumento.Length != 8)
            {
                throw new Exception("El número de documento debe contener 8
dígitos, incluido el verificador");
            }
        }

    }

    public void ValidarFidelizacion()
    {
        if (NivelFidelizacion <1 || NivelFidelizacion > 4)
        {
            throw new Exception("El nivel de fidelización debe ser un valor
entre 1 y 4");
        }
    }
```

// IComparable.cs : ----------

```
  public interface
IComparable
    {
        int CompareTo(object obj);
    }
```

// IValidar.cs : ----------

```
  public interface
IValidar
    {
        void ValidarEstado();
    }
```

// Operador.cs : ----------
--

```
public class Operador : Usuario
{
    private string nombre; public string Nombre { get => nombre; }
private string apellido; public string Apellido { get => apellido; }
    private DateTime fechaIngresoTrabajo; public DateTime
FechaIngresoTrabajo { get => fechaIngresoTrabajo; }
```

```
    public Operador(string email, string contraseña, string nombre, string
apellido, DateTime fechaIngresoTrabajo) : base(email, contraseña)
    {
        this.nombre = nombre;
this.apellido = apellido;
        this.fechaIngresoTrabajo = fechaIngresoTrabajo;
    }



}
```

// Proveedor.cs : ----------

```
public class Proveedor : IValidar, IComparable<Proveedor>,
IEquatable<Proveedor>
    {

        private string nombreProveedor; public string NombreProveedor { get
=> nombreProveedor; }

        private string telefono; public string Telefono { get => telefono; }

        private string direccion; public string Direccion { get => direccion;
}

        private decimal descuento; public decimal Descuento { get =>
descuento; set => descuento = value; }

        public Proveedor(string nombreProveedor, string telefono, string
direccion, decimal descuento)
        {
            this.nombreProveedor = nombreProveedor;
this.telefono = telefono;             this.direccion
= direccion;             this.Descuento = descuento;
        }

        public void ValidarEstado()
        {
            ValidarNombreProveedor();
            ValidarTelefono();
            ValidarDireccion();
        }
        public void ValidarNombreProveedor()
        {
            if (string.IsNullOrEmpty(NombreProveedor))
            {
                throw new Exception("El campo Nombre no puede ser vacío");
            }
        }
        public void ValidarTelefono()
        {
            if (string.IsNullOrEmpty(Telefono))
            {
                throw new Exception("El campo Teléfono no puede ser vacío");
            }
        }
        public void ValidarDireccion()
```

```
        {
            if (string.IsNullOrEmpty(Direccion))
            {
                throw new Exception("El campo Dirección no puede ser
vacío");             }
        }
        public override string ToString()
        {
            return $"Nombre: {NombreProveedor}\n" +
$"Teléfono: {Telefono}\n" +
                   $"Dirección: {Direccion}\n" +
                   $"Descuento: {Descuento}\n" +
                   $"----------------------\n";
}
        public int CompareTo(Proveedor other)
        {
            return NombreProveedor.CompareTo(other.NombreProveedor);
        }



        public bool Equals(Proveedor other)
        {
            return other != null && NombreProveedor == other.NombreProveedor;
        }
    }
```

// TipoDocumento.cs : ----------

```
  public enum
TipoDocumento
    {
       CI = 0,
       PASAPORTE = 1,
       OTROS = 2,
    }
```

// Usuario.cs : ----------

```
public abstract class Usuario : IValidar
{
    private string email; public string Email {get=>email;}

      private string contraseña; public string Contraseña {get=>
contraseña;}

    public Usuario(string email, string contraseña)
    {
        this.email = email;
this.contraseña = contraseña;
    }
```

```
    public virtual void ValidarEstado()
    {
        ValidarEmail();
        ValidarContraseña();
    }
    public void ValidarEmail()
    {
        if (!(Email.Contains("@") && Email.IndexOf("@") != 0 &&
Email.IndexOf("@") != Email.Length - 1)) //VALIDO QUE SE INGRESE EL "@" Y NO
ESTÉ NI AL PRINCIPIO NI AL FINAL
        {             throw new Exception("El email debe contener el
caracter @ y éste no puede estar ni al principio ni al final");
        }
}
    public void ValidarContraseña() {
if (Contraseña.Length < 8)
        {
            throw new Exception("La contraseña debe tener al menos 8
caracteres");
        }
    }



}
```

Casos de prueba:

Funcionalidad	Datos	Pasos	Resultado esperado	Resultado obtenido
Login		1- Ingresa Email valido 2- Ingresa Contraseña valida. 3- Clic en iniciar sesión	Logueo exitoso, se redirige a pagina principal según el rol.	OK
Login – credenciales invalidas		1- Ingresa Email valido 2- Ingresa Contraseña valida. 3- Clic en iniciar sesión.	El sistema despliega mensaje de credenciales invalidas.	OK
Login – formato de entrada invalido o campos vacíos.		1- Ingresa Email sin “@” 2-No ingresa contraseña 3- Clic en iniciar sesión.	El sistema indica que el formato del campo email no es válido. Los campos vacíos deben completarse.	OK
Visualizar Actividades – Filtrar por fecha.		1-Usuario hace Clic en pestaña “Actividades”. 2- Ingresa una fecha a buscar. 3- Clic en “Buscar”	El sistema devuelve una lista con todas las actividades para esa fecha.	OK

Agendar actividad - Operador		1- Usuario se loguea como operador. 2- Va a “Listado de actividades”. 3- Busca por una actividad y le da clic a “Agendar”.	El sistema redirecciona a la pantalla de Login.	OK
Agendar actividad - Huésped		1- Usuario se loguea como huésped. 2- Va a “Listado de actividades”. 3- Busca por una actividad y le da clic a “Agendar”.	El sistema realiza la agenda y muestra el detalle de la misma.	OK
Registro Huésped – Campos sin completar		1- El usuario ingresa a “Registro Huésped”. 2- completa los datos del formulario pero no todos.	El sistema indicará en el campo que corresponda que debe ser completado para registrar al huésped.	OK
Confirmar Agenda		1- usuario/operador se loguea. 2- selecciona la opción “Listar agendas pendientes de pago”. 3- En una de ellas selecciona “confirmar agenda”.	El sistema devuelve un detalle con la agenda en estado “Confirmada”	OK
Actualizar descuento proveedor.		1- usuario/operador se loguea. 2- selecciona la opción “Listado de proveedores”. 3- En uno de ellos selecciona “Aplicar descuento”. 4 - En la casilla “Descuento” ingresa el nuevo monto. 5- Clic en “Asignar Descuento”.	El sistema redirecciona a la lista de proveedores con el campo Descuento actualizado.	