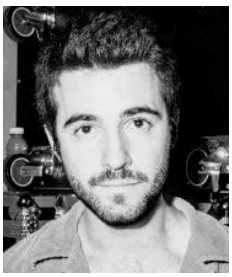
Universidad ORT Uruguay
Facultad de Ingeniería
Escuela de Tecnología

OBLIGATORIO 2 - PROGRAMACIÓN 2

Documentación

[Nicolás Mattos – 149296]



[Nicolás Bañales – 270543]

Grupo: M2A

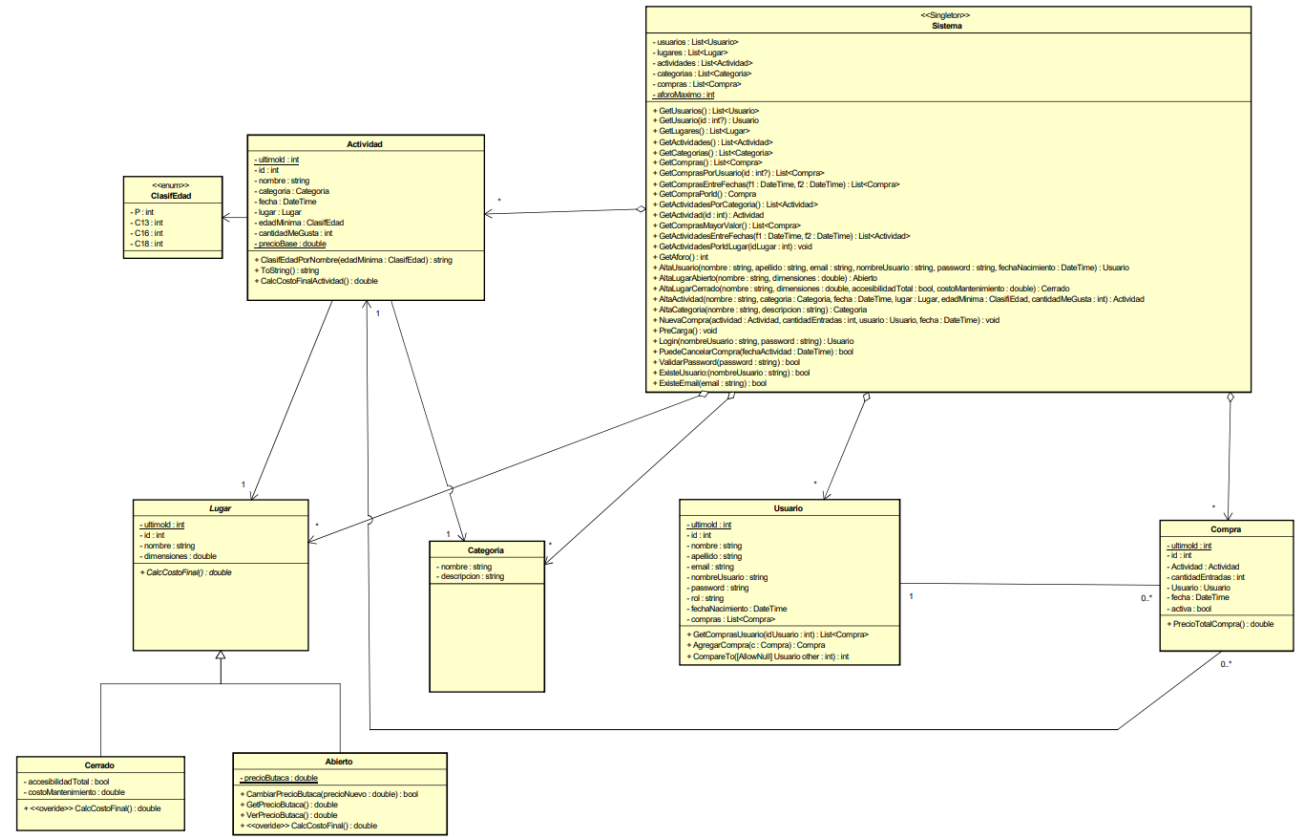
Docente: Joaquín Rodríguez

AP/ATI

Fecha de entrega del obligatorio: 25-11-2021

Índice Documentación

1. Diagrama de clases UML



Para visualizar la imagen mejor adjuntamos el archivo Astah y un PNG en al archivo comprimido.

2. Precarga y Testing

Link sitio online: https://obligatorio-nb-nm.azurewebsites.net/

Otra opción: http://www.obli2nbnm.somee.com/

Recomendación para probar de manera ágil funcionalidades de usuario y operador:

Usuario	Contraseña
eric1 (CLIENTE)	Eric1234
santisape (OPERADOR)	Santi1234

El usuario **eric1** además cuenta con una compra a una actividad a realizarse siempre a menos de 24 horas desde que se **levanta la aplicación** para probar la funcionalidad de cancelación.

El estadio Centenario, no tiene actividades asignadas.

2.1. Datos Precargados

Clientes

Nombre	Apellido	Email	Nombre Usuario	Password	Fecha Nacimiento
Eric	Clapton	eric@gmail.com	eric1	Eric1234	01/01/1969
María	Del Carmen	mary2002@gmail.com	mariac	Mery1234	12/01/2002
Gonzalo	Clapton	gonzag@gmail.com	gonchi22	Gonza1234	01/11/1970
Luciana	Richline	lurichline@gmail.com	luchi99	Lu1234	21/01/1999
George	Richline	george@gmail.com	gmaicol	George1234	21/01/1994
Alberto	Mondongo	alberto@gmail.com	albert3	Alberto1234	21/01/1977

Operadores

Nombre	Apellido	Email	Nombre Usuario	Password	Fecha Nacimiento
Santiago	Ventura	santi@gmail.com	santisape	Santi1234	21/01/1977
Roberto	Bolaño	robert@gmail.com	robertosape	Roberto1234	12/01/1987

Lugares Abiertos

Nombre	Dimensiones (m2)
Parque Rodó	200000
Teatro de Verano	10000
Velódromo	20000
Centenario	22000

Lugares Cerrados

Nombre	Dimensiones(m2)	Accesibilidad Total	Costo Mantenimiento
Teatro Solís	30000	True	30000
Museo Blanes	10000	True	10000
Museo Zorrilla	30000	False	15000

Categorías

Nombre	Descripción
Cine	Vea una película en una pantalla
Teatro	Vea actores en un escenario hacer una obra
Concierto	Vea un espectáculo con músicos en un escenario
Feria	Feria con muchos food trucks y exposiciones

Actividades en lugares abiertos

Nombre	Categoría	Fecha	Lugar	Edad	Cantidad MG
Feria de juegos para todos	Feria	11/11/2021	Parque Rodó	P	5
Agarrate Catalina en Vivo	Concierto	11/02/2022	Teatro de Verano	P	6
Peyote Asesino en Vivo	Concierto	17/11/2022	Velódromo	C18	7
Hamlet - Shakespeare	Teatro	18/11/2022	Teatro de Verano	C16	7
Shakespeare in Love	Cine	19/12/2022	Parque Rodó	C13	7

Actividades en lugares cerrados

Nombre	Categoría	Fecha	Lugar	Edad
Exposición de Picasso Interactiva	Feria	11/11/2021	Museo Blanes	P
Shakespeare – Romeo y Julieta	Teatro	11/11/2022	Teatro Solís	P
Feria Interior de comida	Feria	17/11/2022	Museo Zorrilla	P
Concierto para violín	Concierto	18/11/2022	Teatro Solís	P
Feria esotérica	Feria	19/12/2022	Museo Zorrilla	C13
Fecha Mañana Evento	Feria	Mañana	Museo Zorrilla	C13

Compras

Actividad	Cantidad Entradas	Usuario	Fecha Compra	Activa
Concierto para violín	8	eric1	17/01/2021	True
Agarrate Catalina en Vivo	1	mariac	15/03/2021	True
Concierto para violín	8	gonchi22	11/04/2021	True
Shakespeare in Love	7	luchi99	04/05/2021	True
Fecha Mañana Evento	2	eric1	08/05/2021	True

2.2. Casos de prueba

¿Qué estoy probando?	Resultado Esperado	¿Con que prueba?	¿Pasa la prueba?
Visualización de página por usuario anónimo	Poder ver las actividades disponibles, login y registrarse en el sitio	Entrando a la página anónimamente	SI
Formulario Registro, Todos los Campos	Ver aviso de error al dejar cualquiera de los campos vacíos en su correspondiente	Dejando cualquiera de los campos vacíos	SI
Formulario Registro, Nombre	Probar validación de Nombre ingresado	Completar el campo Nombre con menos de 2 caracteres	NO

Formulario Registro, Nombre	Probar validación de Nombre ingresado	Completar el campo Nombre con 2 o más letras en minúsculas y/o mayúsculas	SI
Formulario Registro, Nombre	Probar validación de Nombre ingresado	Completar el campo Nombre con 2 o más letras y algún numero	NO
Formulario Registro, Usuario	Ver aviso de error al registrar un usuario ya registrado anteriormente	Escribir el campo Usuario con un usuario ya registrado anteriormente.	SI
Formulario Registro, Usuario	Probar validación de Usuario ingresado	Escribir el campo Usuario con menos de 4 caracteres	NO
Formulario Registro, Usuario	Probar validación de Usuario ingresado	Escribir el campo Usuario con 4 o más caracteres	SI
Formulario Registro, Usuario	Probar validación de Usuario ingresado	Escribir el campo Usuario con 4 caracteres o más y algún numero	SI
Formulario Registro, Contraseña	Probar validación de Contraseña ingresada	Escribir en el campo Contraseña: 1 may, 1 min., 1 núm. y completar 6 o mas caracteres	SI
Formulario Registro, Contraseña	Probar validación de Contraseña ingresada	Escribir en el campo Contraseña: 1 may, 1 min., 1 núm. y menos de 6 caracteres	NO
Formulario Registro, Contraseña	Probar validación de Contraseña ingresada	Escribir en el campo Contraseña al menos 6 caracteres pero sin may, min, o num.	NO
Formulario Registro, Confirmar Contraseña	Probar validación de Confirmar Contraseña	Escribir en el campo Confirmar Contraseña, una contraseña diferente a la del campo anterior	NO
Formulario Registro, Confirmar Contraseña	Probar validación de Confirmar Contraseña	Escribir en el campo Confirmar Contraseña, una contraseña igual a la del campo anterior	SI
Formulario Registro, Email	Probar validación Email	Escribir un email sin el formato correcto	NO
Formulario Registro, Email	Probar validación Email	Escribir un email con el formato correcto	SI
Formulario Registro, Email	Ver aviso de error al registrar un email ya registrado anteriormente	Escribir el campo Email con un usuario ya registrado anteriormente.	SI
Formulario Registro, Fecha Nacimiento	Ver aviso de error al dejar la fecha vacía	Dejar fecha sin ingresar	SI
Formulario Registro, Fecha Nacimiento	Ver aviso de error al tener menos de 12 años	Ingresar fecha menor a 12	SI

Login o Inicio de Sesión	Probar ingreso de usuario	Usar un usuario que existe, con contraseña incorrecta	NO
Login o Inicio de Sesión	Probar ingreso de usuario	Usar un usuario que existe, con contraseña correcta	SI
Login o Inicio de Sesión	Probar ingreso de usuario	Usar un usuario que no existe, con alguna contraseña correcta	NO
Visualización de página por usuario con rol cliente	Poder ver las actividades disponibles con opción a compra, Sus compras, y la opción de cerrar Sesión	Entrando a la página con un usuario con rol cliente	SI
Posibilidad de Cancelar Compra para el usuario con rol cliente	Poder cancelar una compra realizada de una actividad con más de 24hs para su realización.	Cancelando una Compra de una actividad con más de 24hs para su realización	SI
Posibilidad de Cancelar Compra para el usuario con rol cliente	No poder cancelar una compra realizada de una actividad con menos de 24hs para su realización Avisar al usuario.	Tratar de cancelar una Compra de una actividad con menos de 24hs para su realización	SI
Poder dar “Me gusta” de manera indefinida	Incremento de “Me gusta” en 1 cada vez que se realiza la acción de apretar el botón	Dándole click al botón “Me gusta” en la lista de Actividades	SI
Imposibilidad de ver la página de registro o login una vez ingresado como usuario con rol cliente	Ser redirigido al Inicio	Escribir en la barra de direcciones la página de login o registro	SI
Visualización de página por usuario con rol operador	Poder ver las actividades disponibles, ver las compras realizadas entre dos fechas, ver la lista de clientes registrados, ver las actividades por lugar, ver las actividades por fechas y categoría y ver las compras más caras.	Entrando a la página con un usuario con rol operador	SI
En Filtro Compras entre dos fechas, validar que el	Ver mensaje de error en caso de que se deje alguna fecha vacía	Dejando una fecha vacía en el filtro de compras entre dos fechas	SI

usuario ingrese dos fechas			
En Filtro Compras entre dos fechas, compras sin resultados	Mensaje de aviso en caso de que no haya resultados	Elegir fechas donde sabemos que no hay resultados	SI
Orden alfabético por apellido y luego nombre en la lista de clientes	Que la lista este ordenada como indica la letra del obligatorio, por apellido y en caso del mismo apellido, por nombre	Precargar dos usuarios con el mismo apellido para ver si se ordenan correctamente	SI
Actividades por lugar	Retornar actividades que se realizan en determinado lugar	Seleccionando un lugar del select	SI
Actividades por lugar sin resultados	Mensaje de aviso en caso de que no haya resultados	Seleccionando un lugar del select que sabemos no tiene actividades	SI
Actividades por fecha y categoría	Retornar actividades que se realizan en determinadas fechas y categoría	Llenando el formulario con fechas y categoría	SI
Actividades por fecha y categoría sin resultados	Mensaje de aviso en caso de que no haya resultados	Seleccionando fechas y categoría que sabemos no hay actividades	SI
Ver compras de mayor valor (más de 1)	Ver compras de mayor valor, en caso de que haya más con el mismo valor máximo, mostrarla también	Precargar dos compras con el valor máximo	SI
Ver compra de mayor valor	Ver compra de mayor valor	Precargar una compra con un valor máximo	SI

3. Código fuente

3.1. Clase Sistema

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

namespace Obligatorio_2_NB_NT_V2.Models
{
    public class Sistema
    {
        private static int aforoMaximo = 65;

        #region Singleton

        private static Sistema instancia = null;

        public static Sistema GetInstancia()
        {
            if (instancia == null)
            {
                instancia = new Sistema();
            }
            return instancia;

        }

        private Sistema()
        {
            PreCarga();

        }

        #endregion


        #region Listas
        private List<Usuario> usuarios = new List<Usuario>();
        private List<Lugar> lugares = new List<Lugar>();
        private List<Actividad> actividades = new List<Actividad>();
        private List<Categoria> categorias = new List<Categoria>();
        private List<Compra> compras = new List<Compra>();

        internal double GetTotal(List<Compra> todasLasCompras)
        {
            double total = 0;

            foreach (Compra compra in todasLasCompras)
            {
                total += compra.PrecioTotal;

            }

            return total;
        }
        #endregion

        #region Obtener Listas
```

```
        public List<Usuario> GetUsuarios() => usuarios;
        public List<Lugar> GetLugares() => lugares;
        public List<Actividad> GetActividades() => actividades;

        public List<Categoria> GetCategorias() => categorias;
        public List<Compra> GetCompras() => compras;
        #endregion

        #region Funciones de Altas

        public Usuario AltaUsuario(string nombre, string apellido, string
email, string nombreUsuario, string password, DateTime fechaNacimiento)
        {

            Usuario nuevoUsuario = null;

            if (ExisteUsuario(nombreUsuario))
            {
                throw new Exception("El usuario ya existe, pruebe uno
nuevo");
            }


            if (ExisteEmail(email))
            {
                throw new Exception("El email ya esta asociado a un usuario
registrado, pruebe uno nuevo");
            }

            if (!ExisteUsuario(nombreUsuario) && !ExisteEmail(email) &&
nombre != "" && nombre != null && nombre.Length >= 2 && apellido.Length >= 2
&& apellido != "" && apellido != null && nombreUsuario.Length >= 4 &&
nombreUsuario != "" && email != "" && email != null && nombreUsuario != ""
&& nombreUsuario != null && fechaNacimiento < DateTime.Now &&
ValidarPassword(password))
            {
                nombreUsuario = nombreUsuario.ToLower();
                nuevoUsuario = new Usuario(nombre, apellido, email,
nombreUsuario, password, fechaNacimiento);
                usuarios.Add(nuevoUsuario);
            }

            return nuevoUsuario;
        }




        public Abierto AltaLugarAbierto(string nombre, double dimensiones)
        {


            Abierto nuevoLugarAbierto = null;

            if (nombre != "" && dimensiones > 0)
            {
                nuevoLugarAbierto = new Abierto(nombre, dimensiones);
                lugares.Add(nuevoLugarAbierto);
            }
            return nuevoLugarAbierto;
```

```
        }

        public Cerrado AltaLugarCerrado(string nombre, double dimensiones,
bool accesoTot, double costoMantenimieno)
        {


                Cerrado nuevoLugarCerrado = null;

                if (nombre != "" && dimensiones > 0 && costoMantenimieno >=
0)
                {
                    nuevoLugarCerrado = new Cerrado(nombre, dimensiones,
accesoTot, costoMantenimieno);
                    lugares.Add(nuevoLugarCerrado);
                }
                return nuevoLugarCerrado;


        }

        public Actividad AltaActividad(string nombre, Categoria categoria,
DateTime fecha, Lugar lugar, Actividad.ClasifEdad edad)
        {

            Actividad nuevaActividad = null;

            if (nombre != "" && categoria != null && fecha != null && fecha
>= DateTime.Now && lugar != null)
            {
                nuevaActividad = new Actividad(nombre, categoria, fecha,
lugar, edad);
                actividades.Add(nuevaActividad);
            }

            return nuevaActividad;
        }

        public Categoria AltaCategoria(string nombre, string descripcion)
        {

            Categoria nuevaCategoria = null;

            if (nombre != "" && descripcion != "")
            {
                nuevaCategoria = new Categoria(nombre, descripcion);
                categorias.Add(nuevaCategoria);
            }

            return nuevaCategoria;


        }

        public Compra NuevaCompra(Actividad actividad, int cantidadEntradas,
Usuario usuario, DateTime fecha, bool activa)
        {

            Compra nuevaCompra = null;

            if (actividad != null && cantidadEntradas > 0 && usuario != null
&& fecha != null && fecha <= DateTime.Now)
```

```
            {
                nuevaCompra = new Compra(actividad, cantidadEntradas,
usuario, fecha, activa);
                compras.Add(nuevaCompra); // se agrega a las compras del
sistema
                usuario.AgregarCompra(nuevaCompra); // se agrega a su propia
lista de compras

            }

            return nuevaCompra;
        }

        #endregion


private void PreCarga()
        {

            // Clientes
            Usuario user1 = AltaUsuario("Eric", "Clapton", "eric@gmail.com",
"eric1", "Eric1234", DateTime.Parse("01-01-1969"));
            Usuario user2 = AltaUsuario("Maria", "Del Carmen",
"mary2002@gmail.com", "mariac", "Mery1234", DateTime.Parse("12-01-2002"));
            Usuario user3 = AltaUsuario("Gonzalo", "Clapton",
"gonzag@gmail.com", "gonchi22", "Gonza1234", DateTime.Parse("01-11-1970"));
            Usuario user4 = AltaUsuario("Luciana", "Richline",
"lurichline@gmail.com", "luchi99", "Lu1234", DateTime.Parse("21-01-1999"));
            Usuario user5 = AltaUsuario("Geogre", "Richline",
"george@gmail.com", "gmaicol", "George1234", DateTime.Parse("21-01-1994"));
            Usuario user6 = AltaUsuario("Alberto", "Mondongo",
"alberto@gmail.com", "albert3", "Alberto1234", DateTime.Parse("21-01-
1977"));

            // Operadores
            Usuario oper1 = AltaUsuario("Santiago", "Ventura",
"santi@gmail.com", "santisape", "Santi1234", DateTime.Parse("21-01-1977"));
            oper1.Rol = "operador";
            Usuario oper2 = AltaUsuario("Roberto", "Bolaño",
"robert@gmail.com", "robertosape", "Roberto1234", DateTime.Parse("12-01-
1987"));
            oper2.Rol = "operador";


            // Lugares Abiertos
            Abierto lugarA1 = AltaLugarAbierto("Parque Rodó", 200000);
            Abierto lugarA2 = AltaLugarAbierto("Teatro de Verano", 10000);
            Abierto lugarA3 = AltaLugarAbierto("Velódromo", 20000);
            Abierto lugarA4 = AltaLugarAbierto("Centenario", 22000);

            // Lugares Cerrados
            Cerrado lugarC1 = AltaLugarCerrado("Teatro Solis", 30000, true,
30000);
            Cerrado lugarC2 = AltaLugarCerrado("Museo Blanes", 10000, true,
10000);
            Cerrado lugarC3 = AltaLugarCerrado("Museo Zorilla de San
Martín", 30000, false, 15000);

            // Categorias
            Categoria categoria1 = AltaCategoria("Cine", "Vea una pelicula
en una pantalla");
```

```
            Categoria categoria2 = AltaCategoria("Teatro", "Vea actores en
un escenario hacer una obra");
            Categoria categoria3 = AltaCategoria("Concierto", "Vea un
espectaculo con músicos en un escenario");
            Categoria categoria4 = AltaCategoria("Feria", "Feria con muchos
food trucks y exposiciones");


            // Actividades en lugares abiertos
            Actividad actividad1 = AltaActividad("Feria de juegos para
todos", categoria4, DateTime.Parse("21-12-2021"), lugarA1,
Actividad.ClasifEdad.P);
            Actividad actividad2 = AltaActividad("Agarrate Catalina en
vivo", categoria3, DateTime.Parse("01-02-2022"), lugarA2,
Actividad.ClasifEdad.P);
            Actividad actividad3 = AltaActividad("Peyote Asesino en vivo",
categoria3, DateTime.Parse("17-11-2022"), lugarA3,
Actividad.ClasifEdad.C18);
            Actividad actividad4 = AltaActividad("Hamlet - Shakespeare",
categoria2, DateTime.Parse("18-11-2023"), lugarA2,
Actividad.ClasifEdad.C16);
            Actividad actividad5 = AltaActividad("Matrix 4", categoria1,
DateTime.Parse("19-12-2022"), lugarC1, Actividad.ClasifEdad.C13);

            // Actividades en lugares cerrados
            Actividad actividad6 = AltaActividad("Exposición de Picasso
Interactiva", categoria4, DateTime.Parse("21-12-2021"), lugarC2,
Actividad.ClasifEdad.P);
            Actividad actividad7 = AltaActividad("Shakespeare - Romeo y
Julieta", categoria2, DateTime.Parse("11-11-2022"), lugarC1,
Actividad.ClasifEdad.P);
            Actividad actividad8 = AltaActividad("Feria Interior de Comida",
categoria4, DateTime.Parse("17-11-2022"), lugarC3, Actividad.ClasifEdad.P);
            Actividad actividad9 = AltaActividad("Concierto para Violin",
categoria3, DateTime.Parse("18-11-2022"), lugarC1, Actividad.ClasifEdad.P);
            Actividad actividad10 = AltaActividad("Ferία Esoterica",
categoria4, DateTime.Parse("19-12-2023"), lugarC3,
Actividad.ClasifEdad.C13);
            // Actividad que se realiza siempre mañana para probar
funcionalidad:
            Actividad actividad11 = AltaActividad("Fecha Mañana Evento",
categoria4, DateTime.Now.AddHours(24), lugarC3, Actividad.ClasifEdad.C13);

            // Alta compras
            Compra compra1 = NuevaCompra(actividad9, 8, user1,
DateTime.Parse("01-01-2021"), true);
            Compra compra2 = NuevaCompra(actividad2, 1, user2,
DateTime.Parse("02-03-2021"), true);
            Compra compra3 = NuevaCompra(actividad9, 8, user3,
DateTime.Parse("03-04-2021"), true);
            Compra compra4 = NuevaCompra(actividad5, 7, user4,
DateTime.Parse("01-05-2021"), true);
            Compra compra5 = NuevaCompra(actividad11, 2, user1,
DateTime.Parse("01-05-2021"), true);

        }




#region Funciones
```

```
        public Usuario Login(string nombreUsuario, string password)
        {
            bool encontrado = false;
            Usuario buscado = null;

            foreach (Usuario u in usuarios) if (!encontrado)
                {
                    if (u.NombreUsuario.Equals(nombreUsuario) &&
u.Password.Equals(password))
                    {
                        encontrado = true;
                        buscado = u;
                    }

                }
            return buscado;
        }



        public bool PuedeCancelarCompra(DateTime fechaActividad)
        {
            bool ret = false;

            DateTime fechaActual = DateTime.Now;

            DateTime fechaLimiteParaCancelar = fechaActividad.AddHours(-24);

            if (fechaActual < fechaLimiteParaCancelar)
            {
                ret = true;
            }

            return ret;
        }


        private bool ValidarPassword(string password)
        {
            string mayusculas = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
            string minusculas = "abcdefghijklmnoupqrstuvwxyz";
            string digitos = "0123456789";

            bool tieneMayuscula = false;
            bool tieneMinuscula = false;
            bool tieneDigito = false;

            if (password.Length >= 6)
            {

                foreach (char c in password)
                {

                    if (mayusculas.Contains(c))
                    {
                        tieneMayuscula = true;
                    }

                    if (minusculas.Contains(c))
                    {
                        tieneMinuscula = true;
                    }
```

```
            if (digitos.Contains(c))
            {
                tieneDigito = true;
            }


        }

        if (tieneMayuscula && tieneMinuscula && tieneDigito)
        {
            return true;
        }

        return false;
    }

    return false;
}


public bool ExisteUsuario(string nombreUsuario)
{
    nombreUsuario = nombreUsuario.Trim().ToLower();

    bool ret = false;

    foreach (Usuario u in usuarios)
    {

        if (u.NombreUsuario == nombreUsuario)
        {
            ret = true;
        }
    }

    return ret;

}

public bool ExisteEmail(string email)
{
    email = email.Trim().ToLower();

    bool ret = false;

    foreach (Usuario u in usuarios)
    {

        if (u.Email == email)
        {
            ret = true;
        }
    }

    return ret;

}



public static int GetAforo() => aforoMaximo;
```

```
public Actividad GetActividad(int id)
{
    Actividad actividad = null;


    foreach (Actividad a in actividades)
    {
        if (a.Id.Equals(id))
        {
            actividad = a;
        }
    }
    return actividad;
}

public Usuario GetUsuario(int? id)
{

    Usuario ret = null;

    foreach (Usuario u in usuarios)
    {
        if (u.Id.Equals(id))
        {
            ret = u;
        }
    }
    return ret;
}

public List<Compra> GetComprasPorUsuario(int? id)
{
    List<Compra> ret = new List<Compra>();

    foreach (Usuario u in usuarios)
    {

        if (u.Id.Equals(id))
        {
            ret = u.GetCompras();
        }

    }
    return ret;

}

public List<Usuario> GetClientes()
{
    List<Usuario> clientes = new List<Usuario>();



    foreach (Usuario u in usuarios)
    {
        if (u.Rol == "cliente")
        {
            clientes.Add(u);
        }
    }

    clientes.Sort();
    return clientes;
```

```
        }

        public Compra GetCompraPorId(int id)
        {
            Compra ret = null;

            foreach (Compra c in compras)
            {

                if (c.Id.Equals(id))
                {
                    ret = c;

                }
            }

            return ret;
        }

        internal List<Actividad> GetActividadesPorCategoria(string
categoria, List<Actividad> listaActividades)
        {

            List<Actividad> ret = new List<Actividad>();

            foreach (Actividad a in listaActividades)
            {

                if (a.Categoria.Nombre == categoria)
                {
                    ret.Add(a);
                }
            }

            return ret;
        }

        public List<Compra> GetComprasMayorValor()
        {

            List<Compra> comprasMayorValor = new List<Compra>();

            double max = 0;

            foreach (Compra c in compras)
            {

                if (c.PrecioTotal > max)
                {
                    comprasMayorValor.Clear();
                    max = c.PrecioTotal;
                    comprasMayorValor.Add(c);
                }

                else if (c.PrecioTotal == max)
                {
                    comprasMayorValor.Add(c);

                }

            }

            return comprasMayorValor;
```

```
        }
        internal List<Actividad> GetActividadesEntreFechas(DateTime f1,
DateTime f2)
        {

            List<Actividad> ret = new List<Actividad>();

            // Invertimos fechas en caso de que sea necesario
            if (f1 > f2)
            {
                DateTime aux;
                aux = f2;
                f2 = f1;
                f1 = aux;
            }


            foreach (Actividad a in actividades)
            {

                if (a.Fecha >= f1 && a.Fecha <= f2)
                {
                    ret.Add(a);
                }


            }

            return ret;
        }

        public List<Actividad> GetActividadesPorIdLugar(int idLugar)
        {
            List<Actividad> actividadesEn = new List<Actividad>();

            foreach (Actividad a in actividades)
            {
                if (a.Lugar.Id.Equals(idLugar))
                {
                    actividadesEn.Add(a);

                }
            }

            return actividadesEn;
        }

        internal List<Compra> GetComprasEntreFechas(DateTime f1, DateTime
f2)
        {

            List<Compra> ret = new List<Compra>();

            // Invertimos fechas en caso de que sea necesario
            if (f1 > f2)
            {
                DateTime aux;
                aux = f2;
                f2 = f1;
                f1 = aux;
            }
```

```
        foreach (Compra c in compras)
        {

            if (c.FechaCompra >= f1 && c.FechaCompra <= f2)
            {
                ret.Add(c);
            }


        }

        return ret;
    }
    #endregion
```

3.2. Clase Actividad

```
using System;

namespace Obligatorio_2_NB_NT_V2.Models
{
    public class Actividad
    {

        private static int ultimoId = 1;

        public int Id { get; }

        public string Nombre { get; set; }

        public Categoria Categoria { get; set; }

        public DateTime Fecha { get; set; }

        public Lugar Lugar { get; set; }

        public ClasifEdad EdadMinima { get; set; }

        public int CantidadMeGusta { get; set; }

        public double PrecioFinal { get; set; }

        private static double precioBase = 250;


        public enum ClasifEdad
        {
            P = 0,
            C13 = 13,
            C16 = 16,
            C18 = 18
        }


        public Actividad(string nombre, Categoria categoria, DateTime fecha,
Lugar lugar, ClasifEdad edadMinima)
        {
            Id = ultimoId;
            ultimoId++;
            Nombre = nombre;
            Categoria = categoria;
            Fecha = fecha;
            Lugar = lugar;
            EdadMinima = edadMinima;
            CantidadMeGusta = 0;
            PrecioFinal = CalcCostoFinalActividad();
        }

        public double CalcCostoFinalActividad()
        {
            return Lugar.CalcCostoFinal(precioBase);
        }


        // Metodo que recibe como parametro un enum y retorna un string
explicando la clasificación con un mensaje más amigable para el usuario
        public string ClasifEdadPorNombre(ClasifEdad edadMinima)
        {
```

```
            if ((int)edadMinima == 0)
            {
                return "Para todo público";
            }
            else
            {
                return $"Mayores que {(int)edadMinima}";
            }
          }

        // Redefinimos ToString() para mostrar las caracteristicas de la
Actividad
        public override string ToString()
        {
            return $"Nombre: {Nombre} | Lugar: {Lugar.Nombre} | Fecha:
{Fecha.ToString("dd/MM/yyyy")} | Clasificacion Edad:
{ClasifEdadPorNombre(EdadMinima)}  | Categoría: {Categoria.Nombre}";
        }

    }
}
```

3.3. Clase Categoría

```
public class Categoria
    {
        public string Nombre { get; set; }
        public string Descripcion { get; set; }

        public Categoria(string nombre, string descripcion)
        {
            Nombre = nombre;
            Descripcion = descripcion;
        }
```

3.4. Clase Compra

```
  using System;

namespace Obligatorio_2_NB_NT_V2.Models
{
    public class Compra
    {

        private static int ultimoId = 1;

        public int Id { get; }

        public Actividad Actividad { get; set; }

        public int CantidadEntradas { get; set; }

        public Usuario Usuario { get; set; }

        public DateTime FechaCompra { get; set; }

        public bool Activa { get; set; }

        public double PrecioTotal { get; set; }

        public double PrecioTotalCompra()
        {
            return CantidadEntradas * Actividad.CalcCostoFinalActividad();
        }

        public Compra(Actividad actividad, int cantidadEntradas, Usuario
usuario, DateTime fechaCompra, bool activa)
        {
            Id = ultimoId;
            ultimoId++;
            Actividad = actividad;
            CantidadEntradas = cantidadEntradas;
            Usuario = usuario;
            FechaCompra = fechaCompra;
            Activa = activa;
            PrecioTotal = PrecioTotalCompra();
        }

    }
}
```

3.5. Clase Lugar

```
namespace Obligatorio_2_NB_NT_V2.Models
{
    public abstract class Lugar
    {

        protected static int ultimoId = 1;

        public int Id { get; set; }

        public string Nombre { get; set; }

        public double Dimensiones { get; set; }

        public abstract double CalcCostoFinal( double precioBase);

    }
}
```

3.6. Clase Abierto

```
namespace Obligatorio_2_NB_NT_V2.Models
{
    public class Abierto : Lugar
    {

        private static double precioButaca = 150;

        public Abierto(string nombre, double dimensiones)
        {
            Id = ultimoId;
            ultimoId++;
            Nombre = nombre;
            Dimensiones = dimensiones;
        }

        // Metodo para cambiar el precio de la butaca, toma como parametro
un Double y si se cumple la validación se asigna el nuevo valor, devuleve un
booleano.

        public static bool CambiarPrecioButaca(double precioNuevo)
        {
            bool exito = false;

            if (precioNuevo >= 0)
            {
                precioButaca = precioNuevo;
                exito = true;
            }
            return exito;
        }

        public static double GetPrecioButaca()
        {
            return precioButaca;
        }

        public override double CalcCostoFinal(double precioBase)
        {
            if(Dimensiones > 1000) // Usamos metros cuadrados para nuestra
precarga por lo tanto 1km2 = 1000m2
            {
                return precioBase * 1.10;
            } else
            {
                return precioBase;
            }
        }

    }
}
```

3.7. Clase Cerrado

```
namespace Obligatorio_2_NB_NT_V2.Models
{
    public class Cerrado : Lugar
    {
        public bool AccesibilidadTotal { get; set; }
        public double CostoMantenimiento { get; set; }

        public Cerrado(string nombre, double dimensiones, bool
accesibilidad, double costoMantenimiento)
        {
            Id = ultimoId;
            ultimoId++;
            Nombre = nombre;
            Dimensiones = dimensiones;
            AccesibilidadTotal = accesibilidad;
            CostoMantenimiento = costoMantenimiento;
        }

        public override double CalcCostoFinal(double precioBase)
        {
            double ret = precioBase;


            if(Sistema.GetAforo() < 50)
            {
                ret = ret * 1.30;
            } else if (Sistema.GetAforo() > 50 && Sistema.GetAforo() <= 70)
            {
                return ret = ret * 1.15;
            }


            return ret;

        }

    }
}
```

3.8. Clase Usuario

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;

namespace Obligatorio_2_NB_NT_V2.Models
{
    public class Usuario : IComparable<Usuario>
    {

        private static int ultimoId = 1;

        public int Id { get; set; }

        public string Nombre { get; set; }

        public string Apellido { get; set; }

        public string Email { get; set; }

        public string NombreUsuario { get; set; }

        public string Password { get; set; }

        public string Rol { get; set; }

        public DateTime FechaNacimiento { get; set; }

        private List<Compra> compras = new List<Compra>();


        public List<Compra> GetCompras()
        {
            return compras;
        }

        public void AgregarCompra(Compra c)
        {
            if (c != null)
            {
                compras.Add(c);
            }
        }

        public int CompareTo([AllowNull] Usuario other)
        {
            if (this.Apellido.CompareTo(other.Apellido) > 0)
            {
                return 1;

            }
            else if (this.Apellido.CompareTo(other.Apellido) < 0)
            {
                return -1;
            }
            else
            {
                if (this.Nombre.CompareTo(other.Nombre) > 0)
                {
```

```
                    return 1;
                }
                else if (this.Nombre.CompareTo(other.Nombre) < 0)
                {
                    return -1;
                }
                else
                {
                    return 0;
                }
            }
        }


        // Constructor para Usuario
        public Usuario(string nombre, string apellido, string email, string
nombreUsuario, string password, DateTime fechaNacimiento)
        {
            Id = ultimoId;
            ultimoId++;
            Nombre = nombre;
            Apellido = apellido;
            Email = email;
            NombreUsuario = nombreUsuario;
            Password = password;
            Rol = "cliente";
            FechaNacimiento = fechaNacimiento;
        }

    }
}
```