Al realizar esta prueba de evaluación ACEPTO (individualmente) y ACEPTAMOS (en grupo) la "cláusula de veracidad" por la que no recibiremos ni daremos ayuda en esta prueba y garantizamos la autoría del 100% de los resultados.

El incumplimiento por mi/nuestra parte de los deberes derivados de las buenas prácticas de honestidad académica podrá dar lugar a la adopción de las medidas contenidas en la Normativa de convivencia universitaria y de régimen disciplinario de la Universitat Politècnica de València.

Ejercicio 1 *(5 puntos).* A partir del siguiente enunciado, se pide construir el diagrama de clases en UML, incluyendo los atributos de las clases que creáis oportunos, así como los nombres de las relaciones que detectéis (no es necesario incluir los métodos, ni los tipos de los atributos, ni su visibilidad).

La asociación de casas rurales de España ha solicitado a ISW Soft el diseño de una aplicación móvil que le permita gestionar las reservas a sus casas rurales, así como la administración de las valoraciones de sus clientes. Su intención es poner disponible la aplicación para poder independizarse de las plataformas que actualmente operan mayoritariamente en el mercado.

Para poder pertenecer al catálogo de casas rurales de la aplicación, las casas rurales deberán ser dadas de alta por al menos un propietario de ésta, aunque podrán registrarse más usuarios como propietarios, para poder tener acceso a gestionar las reservas de la casa rural. A todos los propietarios se les exigirá que proporcionen su nombre, su DNI, así como un teléfono de contacto y una dirección de correo electrónico. Además, deberán proporcionar una contraseña, que contenga letras, números y caracteres especiales de al menos 10 caracteres de longitud. El sistema pedirá el correo electrónico la contraseña proporcionada como credencial.

El propietario deberá indicar, en el momento del alta de la casa rural, el nombre de ésta, su dirección completa, la dirección de su sitio web, su categoría y el precio de la casa por noche. También se requerirá que proporcionen la dirección de correo electrónico de contacto de la casa rural, donde los posibles clientes puedan dirigirse para preguntar dudas. Además, la aplicación dispondrá de un catálogo de servicios (ofrecer desayuno, servicio de comida, servicio de cena, sauna, lavandería, wifi, aparcamiento, etc.), de entre los cuales deberá elegir qué servicios ofrece su casa rural e indicar el precio de cada uno de ellos.

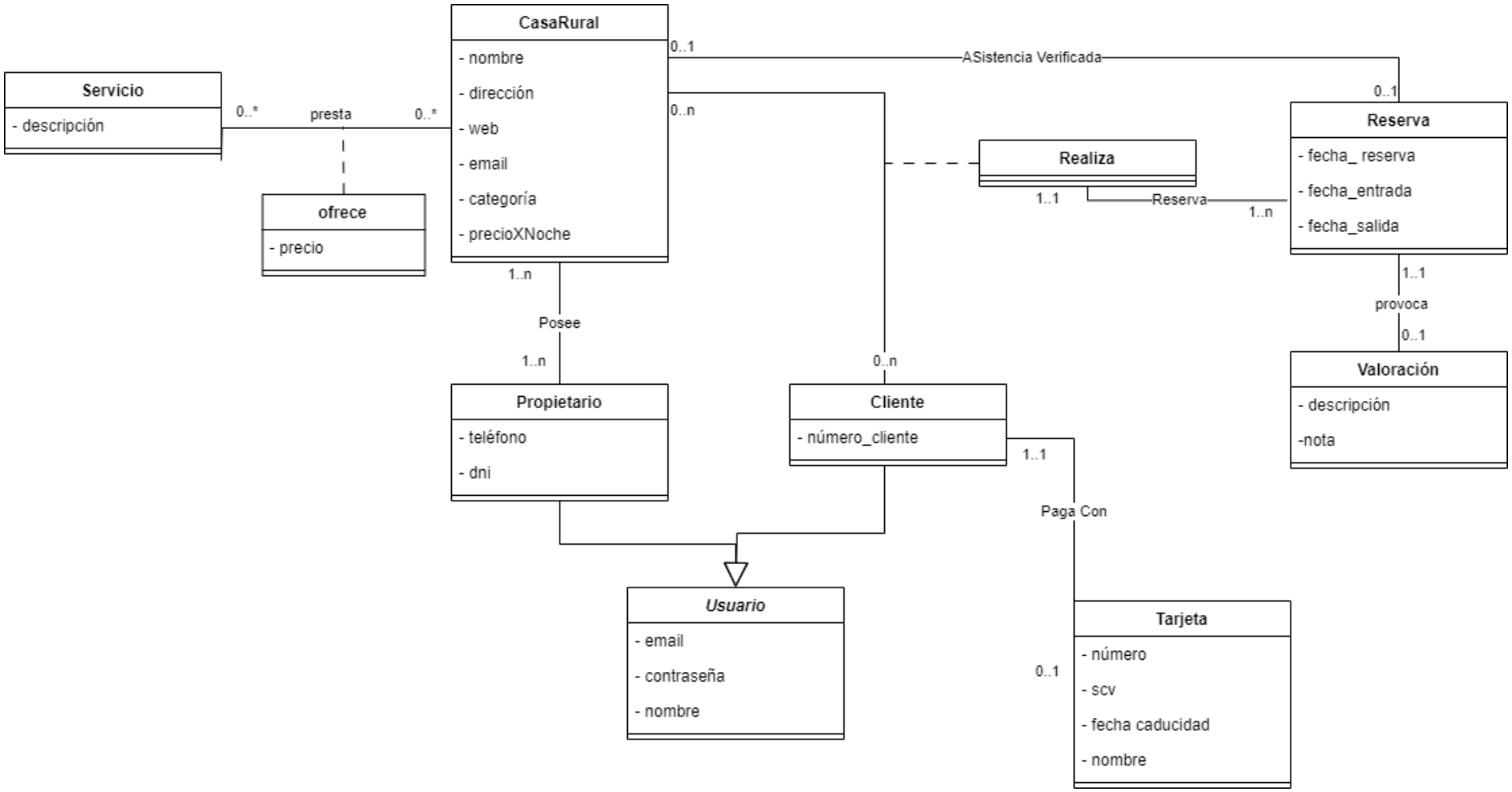
Los clientes podrán acceder libremente al catálogo de casas rurales para ver sus detalles, así como su disponibilidad. Sin embargo, solo podrán realizar reservas y acceder a las valoraciones de otros usuarios si se registran en la aplicación. Para registrarse, los usuarios únicamente deberán proporcionar su nombre, un correo electrónico que les permitirá acceder al sistema, junto con la contraseña que elijan (que contenga letras, números y caracteres especiales de, y tenga al menos 10 caracteres de longitud). Por otro lado, el sistema asignará un número al cliente.

Los clientes podrán reservar cualquier casa rural del catálogo, tantas veces como deseen, indicando la fecha de entrada y salida de la casa rural. El sistema comprobará si el cliente ya había proporcionado con anterioridad los datos de su tarjeta de crédito, puesto que la reserva conlleva un cargo del 5% del importe de la estancia (que será descontado del importe total de la estancia por la casa rural. Si no tiene los datos disponibles, requerirá al cliente que los proporcione (número, SCV, fecha de caducidad y nombre que aparece en la misma), almacenándolos adecuadamente. El cliente podrá modificar los datos de su tarjeta cuando lo desee. Una vez se realice el cargo, se almacenará la reserva con los datos proporcionados por el cliente, junto con la fecha en la que se realiza la reserva.

Una vez los clientes finalizan su estancia en la casa rural, los propietarios estarán obligados a indicar que la reserva ha sido verificada. Tras ello, la aplicación solicitará a los clientes que valoren la estancia en la casa rural, proporcionando un texto de descripción y una nota de 1 a 10. Solo los clientes con reservas verificadas podrán valorar la estancia.

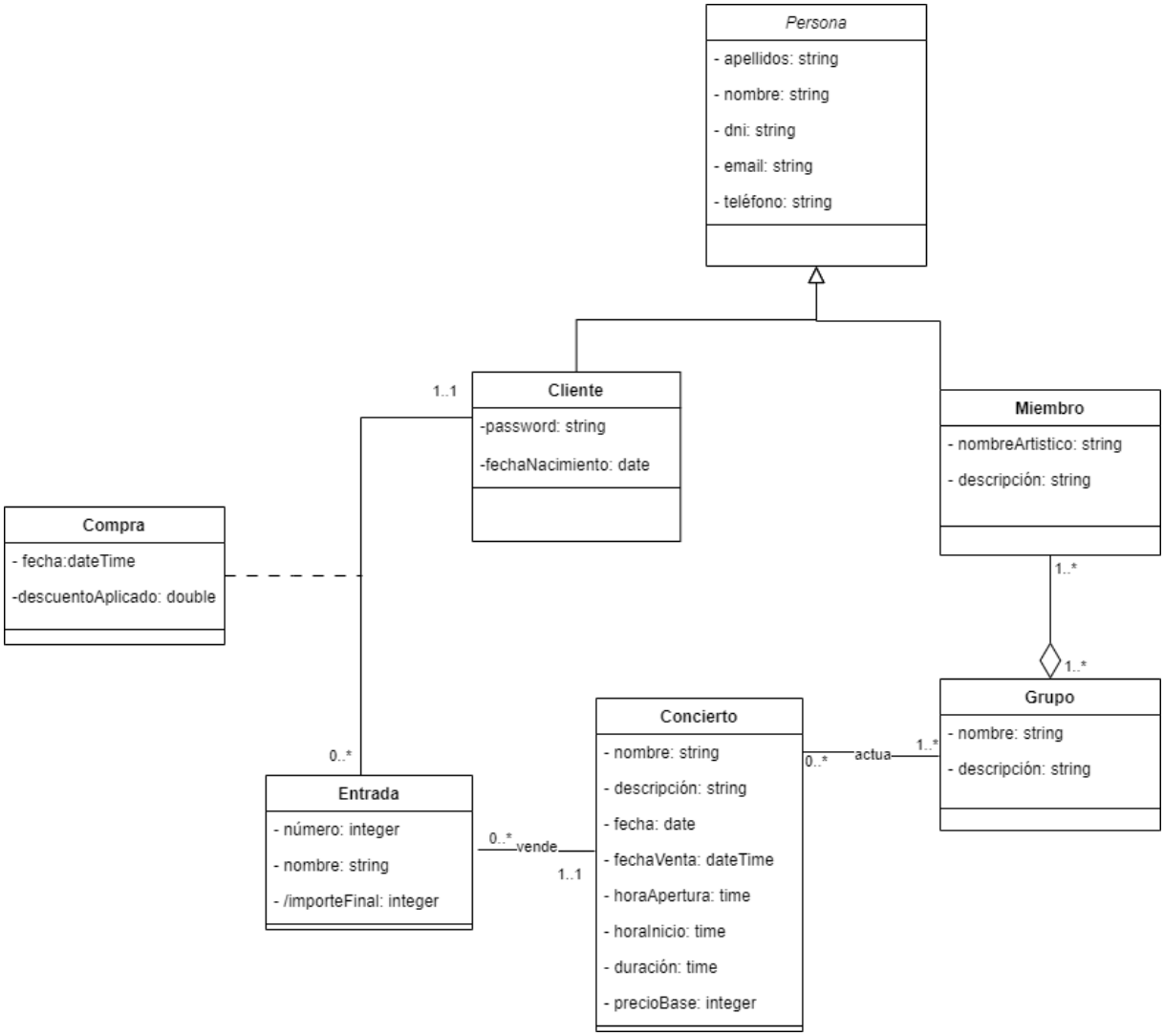
La facturación y cobro de las estancias quedan fuera del ámbito de la aplicación de gestión de reservas.

Solución Ejercicio1:



Ejercicio 2 *(5 puntos)***.** Dado el siguiente diagrama de clases:

a) (2,4 puntos) Realice el diseño en C# de las propiedades de cada clase siguiendo las guías de diseño conocidas. Se deben declarar todos los atributos que se deducen del modelo, pero **no se pide ningún método**.
b) (1,4 puntos) Realice el diseño en C# del constructor con parámetros de cada clase (sólo la cabecera, no los implemente).
c) (1,2 puntos) Usando los constructores del apartado anterior cree los objetos necesarios para crear una instancia del sistema que contenga un grupo musical con un miembro, que dará un concierto del que han vendido una entrada a un cliente.



Solución Ejercicio 2: En esta solución se ha añadido el código de los constructores, que no se pedía.

```
namespace caso1_2024
{
  internal class Program
  {  public abstract class Persona
     {
        private String Apellidos { get; set; }
        private String Nombre { get; set; }
        private String Dni { get; set; }
        private String Email { get; set; }
        private String Telefono { get; set; }
        protected Persona(string apellidos, string nombre, string dni, string email, string telefono)
        {
           Apellidos = apellidos;
           Nombre = nombre;
           Dni = dni;           //Habría que comprobar que es un dato válido y lanzar excepción si no lo es
           Email = email;       //Habría que comprobar que es un dato válido y lanzar excepción si no lo es
           Telefono = telefono;   //Habría que comprobar que es un dato válido y lanzar excepción si no lo es
        }
     }
     public class Cliente: Persona
     {
        private string Password { get; set; }
        private DateTime FechaNacimiento { get; set; }
        //Relaciones
        private ICollection<Compra> compras;
        public void AddCompra(Compra compra)
        {
           if (compra != null)
              compras.Add(compra);

        }

        public Cliente(string apellidos, string nombre, string dni, string email,
              string telefono, string password,
              DateTime fechaNacimiento): base (apellidos, nombre,dni,email,telefono)
        {
           Password = password;
           FechaNacimiento = fechaNacimiento;

           compras = new List<Compra>();

        }
     }
     public class Miembro : Persona
     {

        private String NombreArtistico { get; set; }
        private String Descripcion { get; set; }
        //Relaciones
        private ICollection<Grupo> gruposFormaParte;
        public void AddGrupo(Grupo grupo)
        {
           if (grupo != null)
              gruposFormaParte.Add(grupo);
           else
```

```
            throw new ArgumentNullException(nameof(grupo), "El grupo a añadir no puede ser nulo");
        }
        public Miembro(string apellidos, string nombre, string dni, string email,
            string telefono, string nombreArtistico,
            string descripcion) : base(apellidos, nombre, dni, email, telefono) //mínima 1 a 1 con Grupo, Relajo de ese lado
        {
            NombreArtistico = nombreArtistico;
            Descripcion = descripcion;
            gruposFormaParte = new List<Grupo>();
        }


    }
    public class Entrada
    {
        private int Numero { get; set; }
        private String Nombre { get; set; }
        private double ImporteFinal { get; set; }
        //Relaciones
        private Compra CompraCliente { get; set; }
        private Concierto Concierto { get; set; }
        public void AddCompra(Compra compra)
        {
            this.CompraCliente = compra;
            if (compra.Descuento != 0)
                this.ActualizarImporteFinal();
        }
        public void ActualizarImporteFinal()
        {


            if (this.CompraCliente == null)
                ImporteFinal = this.Concierto.PrecioBase;

            else
                ImporteFinal = (1 - this.CompraCliente.Descuento) * this.Concierto.PrecioBase;


        }

        public Entrada(int numero, string nombre, Concierto concierto) //mínimo 1 a 1 con compra, relajamos de este lado
        {
            Numero = numero;
            Nombre = nombre;
            //Relaciones
            Concierto = concierto;

           ActualizarImporteFinal();
        }
    }
    public class Compra
    {
        private DateTime Fecha { get; set; }
        public double Descuento { get; set; }

        //Relaciones
        private Cliente ClienteCompra { get; set; }
```

```
    private Entrada EntradaCompra { get; set; }


    public Compra(DateTime fecha, double descuento, Cliente cliente, Entrada entrada)
    {
        Fecha = fecha;

        if(descuento >= 0 && descuento <=1 )
            Descuento = descuento;
        else
            throw new ArgumentOutOfRangeException(nameof(descuento), "El descuento debe ser un valor decimal entre 0 y 1");
        //Relaciones
        ClienteCompra = cliente;
        EntradaCompra = entrada;
    }
}
public class Concierto
{
    private String Nombre { get; set; }
    private String Descripcion { get; set; }
    private DateTime Fecha { get; set; }
    private DateTime FechaVenta { get; set; }
    private DateTime HoraApertura { get; set; }
    private DateTime HoraInicio { get; set; }
    private DateTime Duracion { get; set; }
    public double PrecioBase { get; set; }

    //Relaciones
    private ICollection<Entrada> entradasConcierto;
    private ICollection<Grupo> gruposConcierto;
    public void AddGrupo(Grupo grupo)
    {
        if (grupo != null)
            gruposConcierto.Add(grupo);
        else
            throw new ArgumentNullException(nameof(grupo), "El grupo a añadir no puede ser nulo");
    }
    public void AddEntrada(Entrada entrada)
    {
        if (entrada != null)
            entradasConcierto.Add(entrada);
        else
            throw new ArgumentNullException(nameof(entrada), "La entrada a añadir no puede ser nulo");
    }

    public Concierto(string nombre, string descripcion, DateTime fecha, DateTime fechaVenta,
        DateTime horaApertura, DateTime horaInicio, DateTime duracion, double precioBase, Grupo grupo)
    {
        Nombre = nombre;
        Descripcion = descripcion;
        Fecha = fecha;
        FechaVenta = fechaVenta;
        HoraApertura = horaApertura;
        HoraInicio = horaInicio;
        Duracion = duracion;
        PrecioBase = precioBase;
```

```
        entradasConcierto = new List<Entrada>();
        gruposConcierto = new List<Grupo>();
        gruposConcierto.Add(grupo);
    }
}
public class Grupo
{
    private String Nombre { get; set; }
    private String Descripcion { get; set; }
    //Relaciones
    ICollection<Concierto> conciertosGrupo;
    ICollection<Miembro> miembrosGrupo;

    public void AddConcierto(Concierto concierto)
    {
        if (concierto != null)
            conciertosGrupo.Add(concierto);
        else
            throw new ArgumentNullException(nameof(concierto), "El concierto a añadir no puede ser nulo");


    }
    public void AddMiembro(Miembro miembro)
    {
        if (miembro != null)
            miembrosGrupo.Add(miembro);
        else
            throw new ArgumentNullException(nameof(miembro), "El miembro a añadir no puede ser nulo");


    }

    public Grupo(string nombre, string descripcion, Miembro miembro)
    {
        Nombre = nombre;
        Descripcion = descripcion;
        conciertosGrupo = new List<Concierto>();
        miembrosGrupo = new List<Miembro>();
        miembrosGrupo.Add(miembro);
    }
}
static void Main(string[] args)
{
    Miembro cantante = new Miembro("García", "Manolo", "37567473D", "manoloGarcia@gmail.com",
        "+3456789678","Manolo García", "Cantante y compositor");
    Grupo grupo = new Grupo("Manolo García y su banda", "´Manolo García en directo", cantante);
    //aseguramos min 1 a 1 relajado en Miembro
    cantante.AddGrupo(grupo);

    DateTime fechaConcierto = new DateTime(2024, 10, 24, 22, 30, 0);
    Concierto concierto = new Concierto("ManoloTour Valencia", "Concierto de Manolo García en Valencia", fechaConcierto ,
                fechaConcierto.AddDays(-31), fechaConcierto.AddMinutes(-60), fechaConcierto,
                fechaConcierto.AddMinutes(120), 25, grupo);
    //Cumplimos el modelo
    grupo.AddConcierto(concierto);
    int numEntradas = 0;
```

```
        Entrada entrada = new Entrada(++numEntradas, "Soledad Valero", concierto);
        //Cumplimos el modelo
        concierto.AddEntrada(entrada);

        Cliente cliente = new Cliente("Valero", "Soledad", "74036884E", "svalero@dsic.upv.es",
           "6574567", "master@ticket", new DateTime(1977, 02, 10));

        Compra compra = new Compra(DateTime.Now,0.1, cliente, entrada);
        //aseguramos min 1 a 1
        entrada.AddCompra(compra);

        //Cumplimos el modelo
        cliente.AddCompra(compra);

        Console.WriteLine("Instancia del modelo creada. Pulse intro para terminar");
        Console.ReadLine();


    }
  }
}
```

Criterios Corrección

- Ejercicio 1: 5 puntos
  - Dimensión: 6 clases importantes**, 1 especializaciones, 2 relaciones con atributo, 4 relaciones simples
  - Puntuación
    - -0,25 por relación de especialización no detectada
    - -0,3 por relación con atributo no detectada *
    - -0,4 por clase errónea o que falte
    - -0,25 por relación simple errónea o que falte
    - -0,1 por atributo, cardinalidad o nombre de relación
- Ejercicio 2: 5 puntos
  - -0.3 Clase incorrecta
  - -0.3 si persona o concierto incorrecto
  - -0.2 constructor incorrecto
  - -0.1 por instrucción incorrecta o que falte en la instanciación
  -
- * Realiza puede implementarse como una relación entre casa rural y reserva, y reserva con casa rural
- ** Tarjeta podría incluirse en cliente.