

# Boletín de ejercicios resueltos Tema 5

DISEÑO DE CLASES

ANTONIO GARRIDO;JOAN JOSEP FONS;ELISEO MARZAL;SOLEDAD  
VALERO



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



Escuela Técnica  
Superior de Ingeniería  
Informática

# Boletín de ejercicios resueltos Tema 5

## Contenido

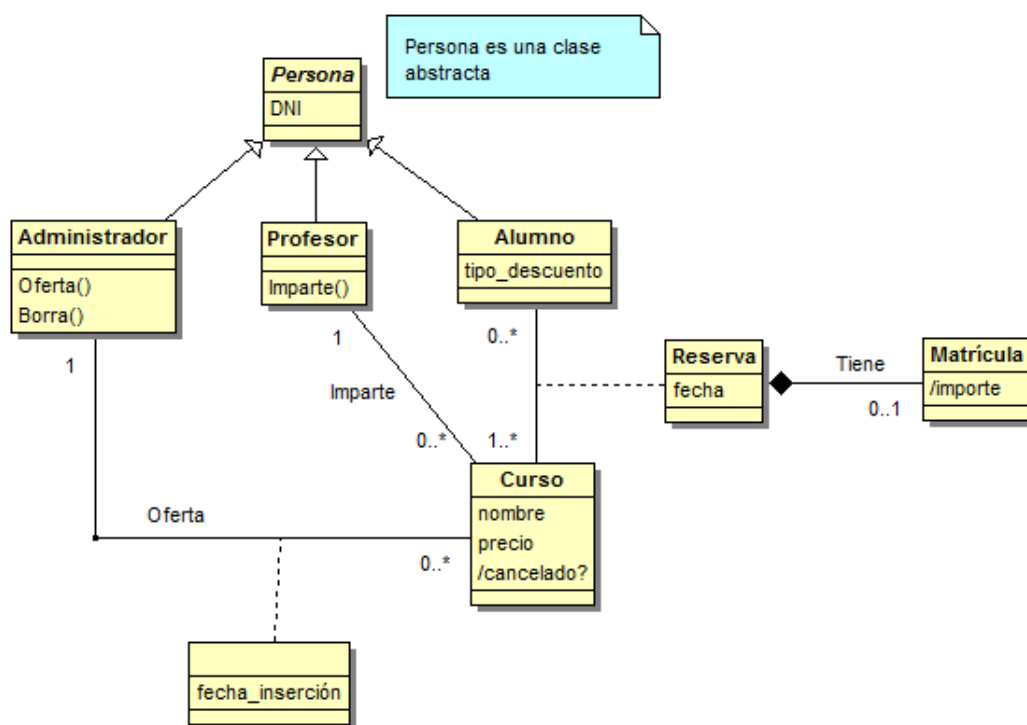
<b>NOTA IMPORTANTE .....</b>	<b>2</b>
1. Escuela de verano.....	2
2. Concesionario de coches .....	4
3. Aplicación de mensajería.....	7
4. Aplicación Academia.....	10
5. Escudería .....	13
6. Clínica veterinaria .....	14
7. Alertas meteorológicas.....	17
8. Aparcamiento .....	19

## NOTA IMPORTANTE

En este boletín se ofrecen resultados de ejercicios de diseño que plantean una solución a una colección de problemas. Para cada ejercicio se ofrece un diseño solución, pero dicha solución no es única y, muy posiblemente, existirán otras soluciones válidas.

### 1. Escuela de verano

Dado el siguiente diagrama de clases en UML, realizad el diseño de clases en C# incluyendo todas las clases, atributos y métodos necesarios. Diseñad también los constructores.



```
public abstract class Persona
{
    private string DNI;

    public Persona(string dni)
    {
        this.DNI = dni;
    }
}
```

```
public class Administrador : Persona
{
    private ICollection<Curso> oferta;

    public void Oferta() { }
```

```

        public void Borra() { }

        public Administrador(string dni) : base(dni)
        {
            oferta = new List<Curso>();
        }
    }

    public class Profesor : Persona
    {
        private ICollection<Curso> imparte;
        public void Imparte() { }

        public Profesor(string dni) : base(dni)
        {
            this.imparte = new List<Curso>();
        }
    }
}

```

```

public class Alumno : Persona
{
    private float tipo_descuento;
    private Dictionary<DateTime, Reserva> reservas;

    public Alumno(string dni, float descuento) : base(dni)
    {
        // por cardinalidad mínima, un alumno debe tener una reserva, pero
        // SE RELAJA para poder crear la reserva
        this.tipo_descuento = descuento;

        this.reservas = new Dictionary<DateTime, Reserva>();
    }
}

```

```

public class Curso
{
    private string nombre;
    private float precio;
    private Administrador oferta;
    private Profesor imparte;
    private DateTime fecha_insercion;
    private Dictionary<DateTime, Reserva> reservas;

    // atributo derivado es un método
    public bool cancelado() { return false; }

    public Curso(string nom, float pre, Administrador admin, Profesor prof,
DateTime fecha)
    {
        this.nombre = nom;
        this.precio = pre;
        this.oferta = admin;
        this.imparte = prof;
        this.fecha_insercion = fecha;
        this.reservas = new Dictionary<DateTime, Reserva>();
    }
}

```

```

public class Reserva
{
    private Alumno alumno;
    private Curso curso;
    private DateTime fecha;
    private Matricula tiene;

    public Reserva(Alumno al, Curso cur, DateTime fe)
    {
        // estamos en una clase asociación y referenciamos tanto al alumno
        // como al curso. Lo podemos hacer porque LO HEMOS RELAJADO en Alumno
        this.alumno = al;
        this.curso = cur;
        this.fecha = fe;
        // this.tiene puede estar vacío, así que podemos inicializarlo a null
    }
}

```

```

public class Matricula
{
    private Reserva tiene;

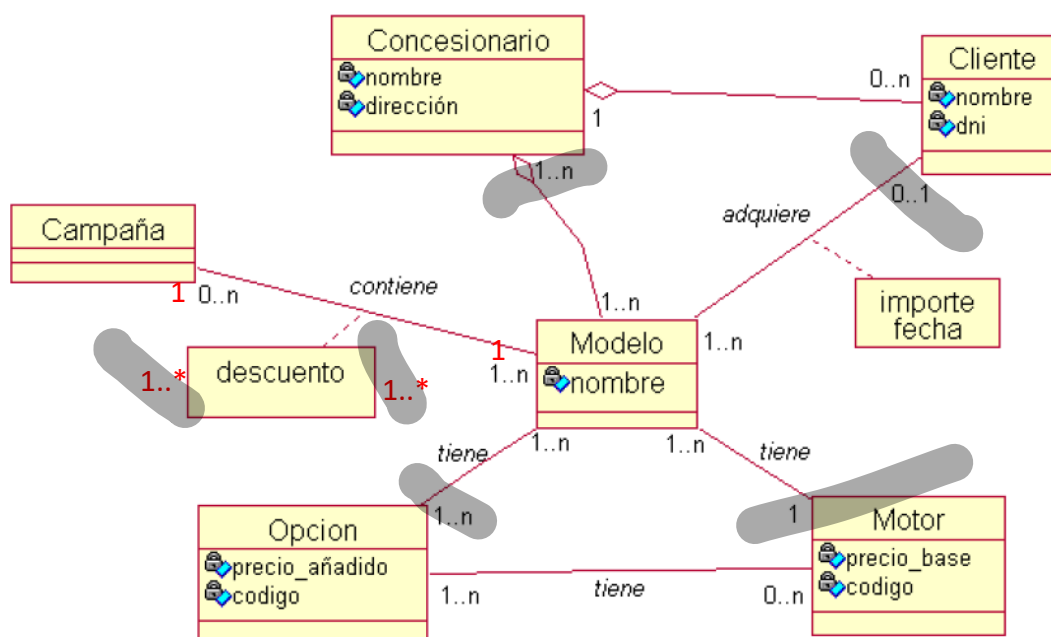
    // atributo derivado es un método
    public float importe() { return 0; }

    public Matricula(Reserva re)
    {
        this.tiene = re;
    }
}

```

## 2. Concesionario de coches

Dado el siguiente diagrama de clases en UML, realizad el diseño de clases en C# incluyendo todas las clases, atributos y métodos necesarios. Diseñad también los constructores.



```

public class Concesionario {
    private string nombre;
    private string direccion;

    private ICollection <Cliente> clientes; //Tenemos una cardinalidad máxima
de N, necesitamos colección
    private ICollection <Modelo> modelos; //Tenemos una cardinalidad máxima
de N, necesitamos colección

    //Tenemos una cardinalidad mínima 1 a 1 con Modelo, relajamos en este lado
y eliminamos Modelo del constructor
    public Concesionario(string nombre, string direccion){
        this.nombre = nombre;
        this.direccion = direccion;

        this.clientes = new List<Cliente>(); //cardinalidad mínima de cero,
solo inicializar

        this.modelos = new List<Modelo>();
        //this.modelos.Add(modelo); --> tendrá que asegurarse por código
    }
}

```

```

public class Cliente {
    private string nombre;
    private string dni;

    private ICollection <Modelo> modelos; //Tenemos una cardinalidad máxima de
N, necesitamos colección
    private Concesionario concesionario;

    //Cardinalidad mínima de 1 con concesionario y con modelo, hay que
pasarlos en el constructor
    public Cliente(string nombre, string dni, Concesionario concesionario,
Modelo modelo){
        this.nombre = nombre;
        this.dni = dni;
        this.concesionario = concesionario;

        this.modelos = new List<Modelo>();
        this.modelos.Add(modelo);
    }
}

```

```

public class Modelo {
    private string nombre;

    private ICollection <Concesionario> concesionarios; //Tenemos una
cardinalidad máxima de N, necesitamos colección
    private ICollection <Contiene> contiene; //El atributo de enlace en una
relación N a N promociona a clase
    private ICollection <Opcion> opciones; //Tenemos una cardinalidad máxima
de N, necesitamos colección
    private Cliente cliente; //cardinalidad máxima 1, mínima cero, no hay que
pasarlo en el constructor
    private Motor motor;

    private double importe; //atributo de la relación con cliente, se pone en
el lado de muchos

```

```

        private DateTime fecha; //atributo de la relación con cliente, se pone en
el lado de muchos

        //Cardinalidad mínima de 1 a 1 con concesionario, lo pasamos al
constructor y relajamos al otro lado
        //Cardinalidad mínima de 1 a 1 con Motor, lo pasamos al constructor y
relajamos al otro lado
        //Cardinalidad mínima de a 1 a 1 con Opcion, lo pasamos al constructor y
relajamos al otro lado
        public Modelo(string nombre, double importe, DateTime fecha, Concesionario
concesionario, Motor motor, Opcion opcion){
            this.nombre = nombre;
            this.importe = importe;
            this.fecha = fecha;

            this.concesionarios = new List<Concesionario>();
            this.concesionarios.Add(concesionario);

            this.contienes = new List<Contiene>(); //cardinalidad mínima de
cero, solo inicializar la colección

            this.opciones = new List<Opcion>();
            this.opciones.Add(opcion);

            this.motor = motor;
        }
}

```

```

public class Contiene {
    //Esta clase surge al promocionar un atributo de una relación N a N a
clase
    private double descuento;

    private Modelo modelo;
    private Campanya campanya;

    public Contiene(double descuento, Modelo modelo, Campanya campanya){
        this.descuento = descuento;
        this.modelo = modelo;
        this.campanya = campanya;
    }
}

```

```

public class Campanya {

    private ICollection <Contiene> contienes; //El atributo de enlace en una
relación N a N promociona a clase

    public Campanya(){
        // La referencia a contiene se ha relajado
        this.contienes = new List <Contiene>();
    }
}

```

```

public class Opcion {
    private double precio_anyadido;
    private int codigo;
    private ICollection<Modelo> modelos; //Tenemos una cardinalidad máxima de
N, necesitamos colección
    private ICollection<Motor> motores; //Tenemos una cardinalidad máxima de N,
necesitamos colección
}

```

```

        //Cardinalidad mínima de 1 a 1 con modelo, hemos relajado en este lado
        public Opcion(double precio_anyadido,int codigo){
            this.precio_anyadido = precio_anyadido;
            this.codigo = codigo;

            this.modelos = new List<Modelo>();
            //this.modelo.Add(modelo) ---> tendrá que asegurarse por código.
Hemos relajado en este lado

            this.motores = new List<Motor>(); //cardinalidad mínima de cero,
solo inicializar la colección
        }
    }
}

```

```

public class Motor {
    private double precio_base;
    private int codigo;

    private ICollection<Modelo> modelos; //Tenemos una cardinalidad máxima de
N, necesitamos colección
    private ICollection < Opcion> opciones; //Tenemos una cardinalidad máxima
de N, necesitamos colección

    //Cardinalidad mínima de 1 a 1 con Modelo, hemos relajado en este lado
    public Motor(double precio_base, int codigo, Opcion opcion){
        this.precio_base = precio_base;
        this.codigo = codigo;

        this.opciones = new List<Opcion>();
        this.opciones.Add(opcion); //carddinalidad minima de 1

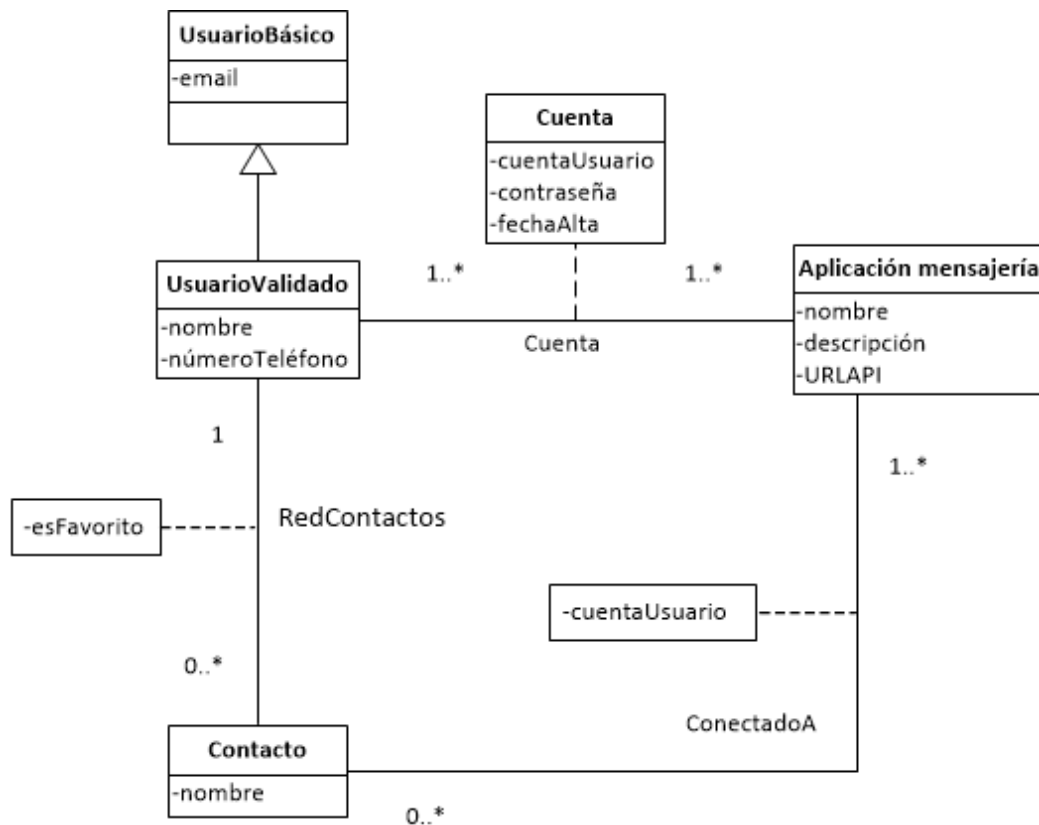
        this.modelos = new List<Modelo>();
        //this.modelos.Add(modelo); --> tendrá que asegurarse por código.
Hemos relajado en este lado
    }
}

```

### 3. Aplicación de mensajería

Dado el siguiente diagrama de clases en UML, realizad el diseño de clases en C# incluyendo todas las clases, atributos y métodos necesarios. Diseñad también los constructores.





```

public class UsuarioBasico
{
    private string email;
    public UsuarioBasico(string email)
    {
        this.email = email;
    }
}

```

```

public class UsuarioValidado : UsuarioBasico
{
    private string nombre;
    private string numeroTelefono;
    // colocar esFavorito aquí sería un ERROR
    private Dictionary<string, Contacto> redContactos;
    // tener una colección de AplicacionMensajería sería un error
    private ICollection<Cuenta> cuentas;

    public UsuarioValidado(string email, string nombre, string numeroTelef) :
base(email)
    {
        this.nombre = nombre;
        this.numeroTelefono = numeroTelef;
        this.redContactos = new Dictionary<string, Contacto>();
        this.cuentas = new List<Cuenta>();
    }
}

```

```

public class Contacto
{
    private string nombre;

    private UsuarioValidado usuarioRedContactos;
    // importante: el atributo de enlace en relación 1-muchos
    // se coloca en la parte de muchos
    private bool esFavorito;
    // tener una colección de AplicacionMensajeria sería un error
    private ICollection<ConectadoA> conectadoA;

    public Contacto(string nombre, UsuarioValidado contacto, bool favorito)
    {
        // ConectadoA se ha relajado
        this.nombre = nombre;
        this.usuarioRedContactos = contacto;
        this.esFavorito = favorito;
        this.conectadoA = new List<ConectadoA>();
    }
}

```

```

public class AplicacionMensajeria
{
    private string nombre;
    private string descripcion;
    private string URLAPI;
    // tener una colección de Contacto sería un error
    private Queue<ConectadoA> conectadoA;
    // tener una colección de UsuarioValidado sería un error
    private ICollection<Cuenta> cuentas;

    public AplicacionMensajeria(string nombre, string desc, string URL)
    {
        this.nombre = nombre;
        this.descripcion = desc;
        this.URLAPI = URL;
        this.conectadoA = new Queue<ConectadoA>();
        this.cuentas = new List<Cuenta>();
    }
}

```

```

public class ConectadoA
{
    private Contacto conectadoAContacto;
    private AplicacionMensajeria conectadoAAplicacion;
    // la clase NO se puede llamar cuentaUsuario
    private string cuentaUsuario;

    public ConectadoA(Contacto contacto, AplicacionMensajeria aplicacion,
string cuenta)
    {
        this.conectadoAContacto = contacto;
        this.conectadoAAplicacion = aplicacion;
        this.cuentaUsuario = cuenta;
    }
}

```

```

public class Cuenta
{
    private string cuentaUsuario;
    private string contraseña;
    private DateTime fechaAlta;

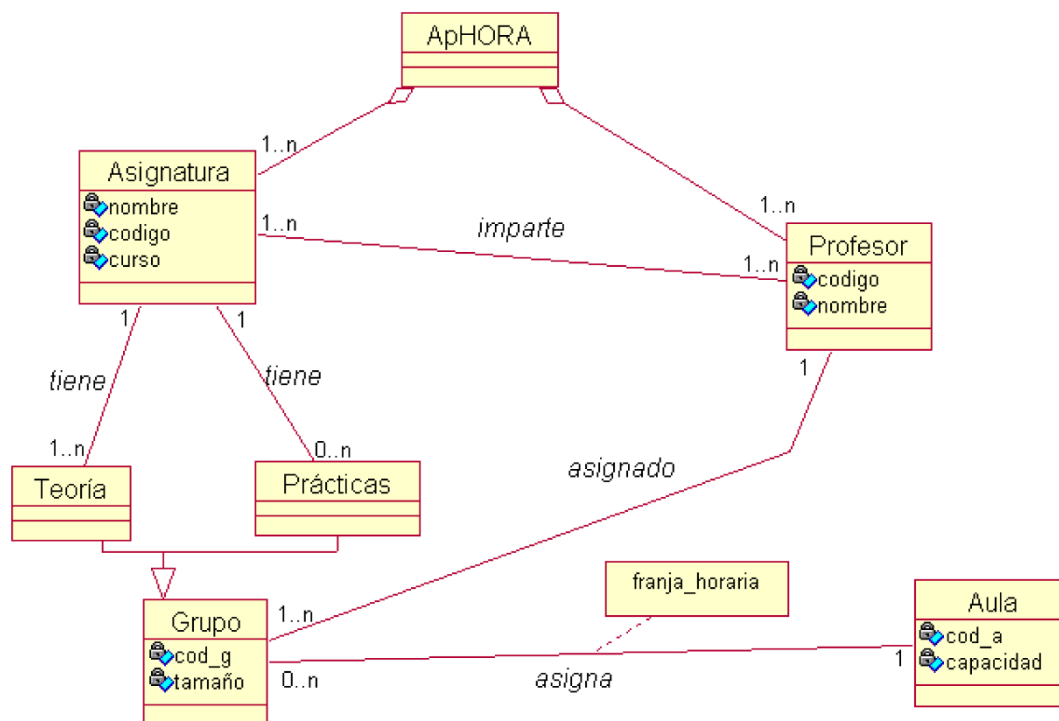
    // la clase asociacion necesita una referencia a las dos
    // clases de la relación, independientemente de la multiplicidad
    private UsuarioValidado usuarioValidado;
    private AplicacionMensajeria aplicacionMensajeria;

    public Cuenta(string cuenta, string cont, DateTime fecha,
        UsuarioValidado usuario, AplicacionMensajeria aplicacion)
    {
        this.cuentaUsuario = cuenta;
        this.contraseña = cont;
        this.fechaAlta = fecha;
        this.usuarioValidado = usuario;
        this.aplicacionMensajeria = aplicacion;
    }
}

```

#### 4. Aplicación Academia

Dado el siguiente diagrama de clases en UML, realizad el diseño de clases en C# incluyendo todas las clases, atributos y métodos necesarios. Diseñad también los constructores.



```

public class ApHora(){
    private ICollection <Profesor> profesores;
    private ICollection <Asignatura> asignaturas;

    //Tenemos una cardinalidad mínima 1 a 1 con Asignatura, relajamos en este lado
    // y eliminamos asignatura del constructor
    //Tenemos una cardinalidad mínima 1 a 1 con Profesor, relajamos en este lado
    // y eliminamos profesor del constructor
    public class ApHora(){
        this.profesores = new List<Profesor>();
        //this.profesores.Add(profesor);--> tendrá que asegurarse por código
        this.asignaturas = new List<Asignatura>();
        //this.asignaturas.Add(asignatura);--> tendrá que asegurarse por código
    }
}

```

```

public class Asignatura{
    private string nombre;
    private int codigo;
    private string curso;
    private ApHora enApHora;

    private ICollection<Profesor> profesores;
    private ICollection<Teoria> gruposTeoria;
    private ICollection<Practicas> gruposPracticas;

    //Tenemos una cardinalidad mínima 1 a 1 con profesor, relajamos en este lado
    // y eliminamos profesor del constructor
    //Tenemos una cardinalidad mínima 1 a 1 con ApHora, relajaremos en ApHora
    //Tenemos una cardinalidad mínima 1 a 1 con Teoria, relajamos en este lado
    // y eliminamos teoria del constructor
    public Asignatura(String nombre, int codigo, String Curso, ApHora apHora){
        this.nombre = nombre;
        this.codigo = codigo;
        this.enApHora = apHora;

        this.profesores = new List<Profesor>();
        //this.profesores.Add(profesor); --> tendrá que asegurarse por código

        this.gruposTeoria = new List<Teoria>();
        //this.gruposTeoria.Add(teoria); --> tendrá que asegurarse por código

        this.gruposPracticas = new List<Practicas>();
    }
}

```

```

public class Profesor {
    private int codigo;
    private string nombre;
    private ApHora enApHora;
    private ICollection <Asignatura> asignaturas;
    private ICollection <Grupo> grupos;

    //Tenemos una cardinalidad mínima 1 a 1 con Grupo, relajamos de este lado y //eliminamos
    grupo del constructor
    //Tenemos una cardinalidad mínima 1 a 1 con ApHora, relajaremos en ApHora
    //Tenemos una cardinalidad mínima 1 a 1 con Asignatura, relajaremos en Asignatura
    public Profesor(int codigo, string nombre, Asignatura asignatura, ApHora
    apHora){
        this.codigo = codigo;
        this.nombre = nombre;
        this.enApHora = apHora;
        this.asignaturas = new List<Asignatura>();
    }
}

```

```

        this.asignaturas.Add(asignatura);
        this.grupos = new List <Grupo>();
        //this.grupos.Add(grupo); --> tendrá que asegurarse por codigo
    }
}

```

```

public class Grupo {
    private int cod_g;
    private int tamanyo;
    private DateTime hora_desde;
    private DateTime hora_hasta;
    private Aula aula;
    private Profesor profesor;

    public Grupo(int cod_g, int tamanyo, DateTime hora_desde, DateTime hora_hasta,
Aula aula, Profesor profesor){
        this.cod_g = cod_g;
        this.tamanyo = tamanyo;
        this.hora_desde = hora_desde;
        this.hora_hasta = hora_hasta;
        this.aula = aula;
        this.profesor = profesor;
    }
}

```

```

public class Aula{
    private int cod_a;
    private int capacidad;
    private ICollection<Grupo> grupos;

    public Aula(int cod_a, int capacidad){
        this.cod_A = cod_a
        this.capacidad = capacidad;
        grupos = new List<Grupo>();
    }
}

```

```

public class Teoria : Grupo {
    private Asignatura asignaturaTeoria;

    public Teoria (int cod_g, int tamanyo, DateTime hora_desde, DateTime hora_hasta,
Aula aula, Profesor profesor, Asignatura asignatura): base (cod_g, tamanyo, hora_desde,
hora_hasta, aula, profesor){

        asignaturaTeoria = asignatura;
    }
}

```

```

public class Practicas : Grupo {
    private Asignatura asignaturaPracticas;

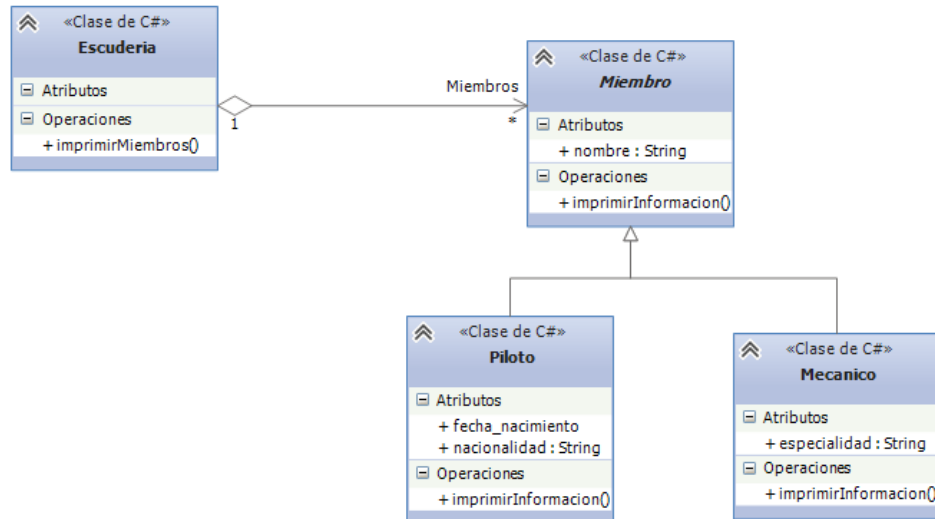
    public Practicas (int cod_g, int tamanyo, DateTime hora_desde, DateTime
hora_hasta, Aula aula, Profesor profesor, Asignatura asignatura): base (cod_g, tamanyo,
hora_desde, hora_hasta, aula, profesor){

        asignaturaPracticas = asignatura;
    }
}

```

## 5. Escudería

Dado el siguiente diagrama de clases en UML, realizad el diseño de clases en C# incluyendo todas las clases, atributos y métodos necesarios. Diseñad también los constructores.



```
public class Escuderia
{
    private ICollection<Miembro> miembros;
    public Escuderia()
    {
        miembros = new List<Miembro>();
    }
}
```

```
public class Miembro
{
    private string nombre;

    public Miembro(String nombre)
    {
        this.nombre = nombre;
    }
}
```

```
public class Piloto : Miembro
{
    private DateTime fecha_nacimiento;
    private string nacionalidad;

    public Piloto(DateTime fecha_nacimiento, string nacionalidad, String nombre)
: base(nombre)
    {
        this.fecha_nacimiento = fecha_nacimiento;
        this.nacionalidad = nacionalidad;
    }
}
```

```
public class Mecanico : Miembro
{
    private string especialidad;
```

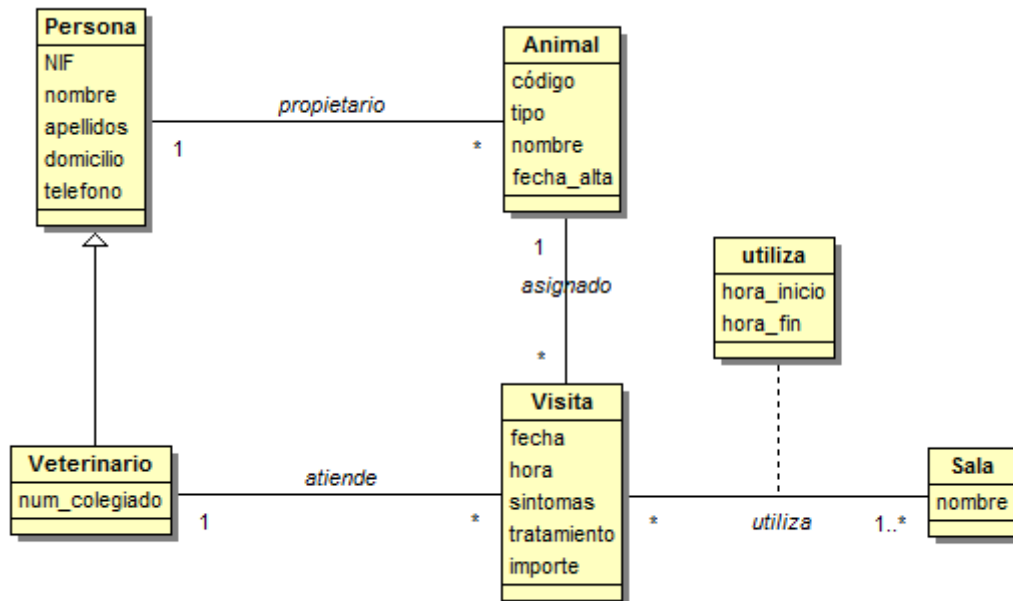
```

public Mecanico(string especialidad, String nombre) : base(nombre)
{
    this.especialidad = especialidad;
}
}

```

## 6. Clínica veterinaria

Dado el siguiente diagrama de clases en UML realizad el diseño en C# incluyendo todas las clases, atributos y constructores.



```

public class Persona
{
    private String NIF;
    private String nombre;
    private String apellidos;
    private String domicilio;
    private String telefono;

    private ICollection<Animal> animales;

    public Persona(String NIF, String nombre, String apellidos, String
        domicilio, String telefono)
    {
        this.NIF = NIF;
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.domicilio = domicilio;
        this.telefono = telefono;

        this.animales = new List<Animal>();
    }
}

```

```

public class Veterinario:Persona
{
    private int num_colegiado;

    public Veterinario(String NIF, String nombre, String apellidos, String
        domicilio, String telefono, int num_colegiado) :
        base(NIF, nombre, apellidos, domicilio, telefono)
    {
        this.num_colegiado = num_colegiado;
    }
}

```

```

public class Animal
{
    private String codigo;
    private String tipo;
    private String nombre;
    private DateTime fecha_alta;

    private Persona propietario;

    private ICollection<Visita> visitasAsignadas;

    public Animal(String codigo, String tipo, String nombre, DateTime
        fecha_alta, Persona propietario)
    {
        this.codigo = codigo;
        this.tipo = tipo;
        this.nombre = nombre;
        this.fecha_alta = fecha_alta;
        this.propietario = propietario;
        this.visitasAsignadas = new List<Visita>();
    }
}

```

```

public class Sala
{
    private String nombre;

    private ICollection<Utiliza> utiliza;

    public Sala(String nombre)
    {
        this.nombre = nombre;
        this.utiliza = new List<Utiliza>();
    }
}

```

```

public class Visita
{
    private DateTime fecha_hora;
    private String sintomas;
    private String tratamiento;
    private decimal importe;

    private Animal asignado;
    private Veterinario atiende;
}

```



```

        private List<Utiliza> utiliza;

        public Visita(DateTime fecha_hora, String sintomas, String tratamiento,
decimal importe, Animal asignado, Veterinario atiende) //, Utiliza utiliza)
        {
            // por cardinalidad mínima, una visita debe tener un utiliza (de una
sala),
            // SE RELAJA para poder crear el objeto Utiliza que requiere de una
visita y una sala
            this.fecha_hora = fecha_hora;
            this.sintomas = sintomas;
            this.tratamiento = tratamiento;
            this.importe = importe;
            this.asignado = asignado;
            this.atiende = atiende;
            this.utiliza = new List<Utiliza>();
            //this.utiliza.Add(utiliza); --> tendrá que asegurarse por código
        }
    }
}

```

```

public class Utiliza
{
    private Visita visita;
    private Sala sala;

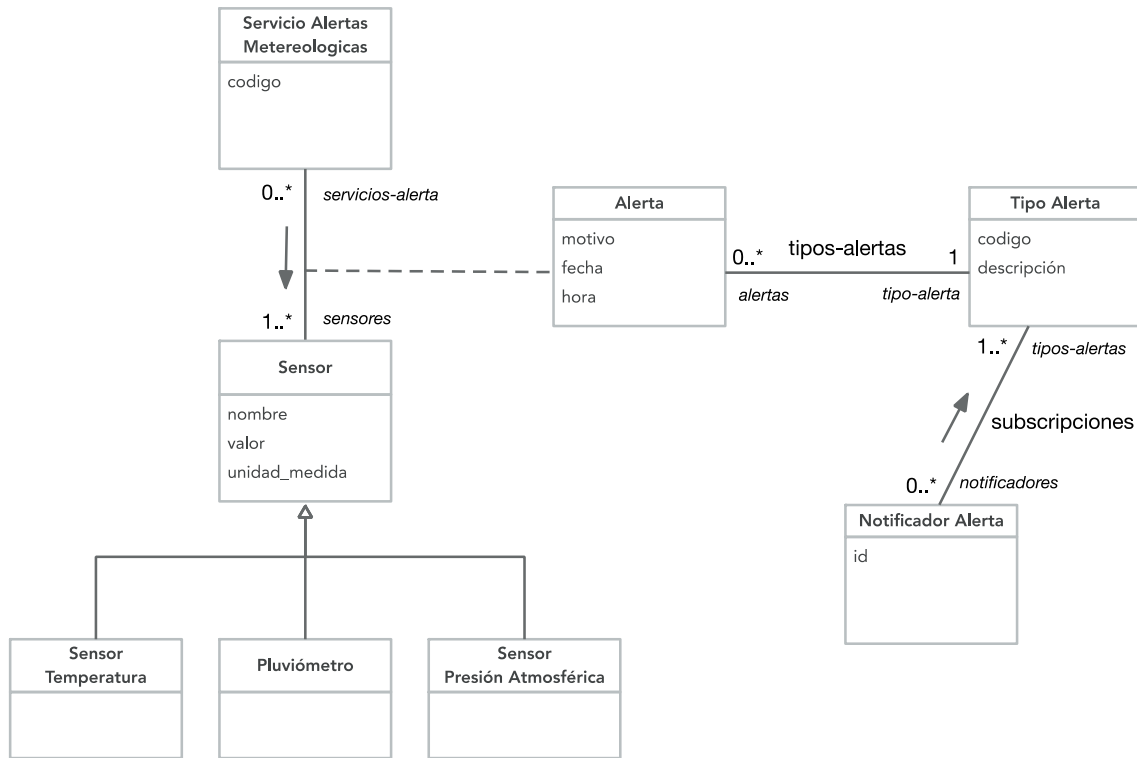
    private DateTime hora_inicio;
    private DateTime hora_fin;

    public Utiliza(Visita visita, Sala sala, DateTime hora_inicio, DateTime
hora_fin)
    {
        // la clase asociación referencia a la visita y a la sala
        // se puede hacer porque SE HA RELAJADO en Visita
        this.visita = visita;
        this.sala = sala;
        this.hora_fin = hora_fin;
        this.hora_inicio = hora_inicio;
    }
}

```

## 7. Alertas meteorológicas

Dado el siguiente diagrama de clases en UML, realizad el diseño de clases en C# incluyendo todas las clases, atributos y métodos necesarios. Diseñad también las cabeceras de los constructores.



**Nota 1.** Hay dos restricciones de navegabilidad definidas (en el sentido de las flechas)

```
public class Servicio_Alertas_Metereologicas {
    public int codigo { get; set; }
    public ICollection<Alerta> alertas { get; set; };

    // Constructor
    public Servicio_Alertas_Metereologicas(int codigo);
    // Relajamos la cardinalidad 1..* con Sensor (Alerta), ya que se genera
    // problema para asignar una Alerta (que también requiere un
    // ServicioAlertasMetereologicas).
    // Habrá que añadir por código la Alerta asociada al servicio, tras ejecutar
    // el constructor
}

public class Sensor {
    public string nombre { get; set; }
    public float valor { get; set; }
    public string unidad_medida { get; set; }
    // La restricción de navegabilidad elimina la colección de Alerta (clase
    // asociación )

    // Constructor
    public abstract Sensor(string codigo, float valor, string unidad_medida);
}

public class Sensor_Temperatura : Sensor {
    // Constructor
}
```

```

        public Sensor_Temperatura(string codigo, float valor, string unidad) :
base(codigo, valor, unidad);
    }

    public class Pluviometro : Sensor {
        // Constructor
        public Pluviometro(string codigo, float valor, string unidad) : base(codigo,
valor, unidad);
    }

    public class Sensor_Presion_Atmosferica : Sensor {
        // Constructor
        public Sensor_Presion_Atmosferica(string codigo, float valor, string unidad)
: base(codigo, valor, unidad);
    }

    public class Alerta {
        public int codigo { get; set; } // Emerge como identificador único
        public string motivo { get; set; }
        public date fecha { get; set; }
        public time hora { get; set; }
        public Servicio_Alertas_Metereologicas servicio_alerta { get; set; }
        public Sensor sensor { get; set; }
        public Tipo_Alerta tipo_alerta { get; set; }

        // Constructor
        public Alerta(int codigo, string motivo, date fecha, time hora,
Sensor sensor, Tipo_Alerta tipo_alerta);
    }

    public class Tipo_Alerta {
        public int codigo { get; set; }
        public string descripcion { get; set; }
        public ICollection<Alerta> alertas { get; set; };
        // La restricción de navegabilidad elimina la colección de Notificador_Alerta

        // Constructor
        public Tipo_Alerta(int codigo, string descripcion);
    }

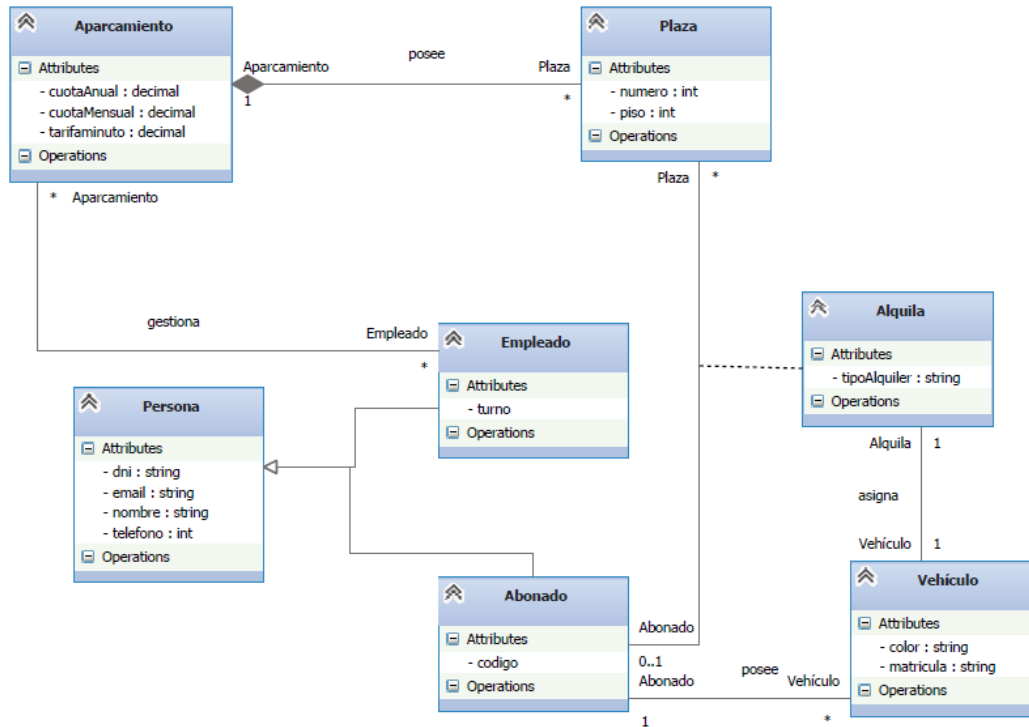
    public class Notificador_Alerta {
        public int id { get; set; }
        public ICollection<Tipo_Alerta> tipos_alertas { get; set; };

        // Constructor
        public Notificador_Alerta(int codigo, Tipo_Alertaa tipo_alerta);
    }

```

## 8. Aparcamiento

Dado el siguiente diagrama de clases, obtener el diseño en C# de todas las clases siguiendo las pautas vistas en clase. Se deben declarar todos los atributos que se deducen del modelo, pero **no se pide ningún método**. También se deben proporcionar las cabeceras de todos los constructores con sus parámetros.



```
public class Aparcamiento
{
    private decimal cuotaAnual;
    private decimal cuotaMensual;
    private decimal tarifaMinuto;

    private ICollection<Plaza> plazas;
    private ICollection<Empleado> empleados;
    public Aparcamiento(decimal cuotaAnual, decimal cuotaMensual, decimal tarifaMinuto)
    {}
}

public class Persona
{
    private string dni;
    private string email;
    private string nombre;
    private int telefono;

    public Persona(string dni, string email, string nombre,int telefono)
    {}
}
```

```

public class Empleado : Persona
{
    private string turno;
    private ICollection<Aparcamiento> aparcamientos;

    public Empleado(string dni, string email, string nombre, int telefono, string turno) :
base( dni, email, nombre, telefono)
    {}
}

public class Abonado : Persona
{
    private int codigo;
    private ICollection<Alquila> alquileres;
    private ICollection<Vehiculo> vehiculos;
    public Abonado(string dni, string email, string nombre, int telefono, int codigo) :
base(dni, email, nombre, telefono)
    {}
}

public class Plaza
{
    private int numero;
    private int piso;
    private Aparcamiento aparcamiento;
    private Alquila alquila;

    public Plaza(int numero, int piso, Aparcamiento aparcamiento)
    {}
}

public class Alquila
{
    private string tipoAlquiler;
    private Plaza plaza;
    private Abonado abonado;
    private Vehiculo vehiculo;

    public Alquila(string tipoAlquiler, Plaza plaza, Abonado abonado, Vehiculo vehiculo)
    { }
}

public class Vehiculo
{
    private string color;
    private string matricula;
    private Abonado abonado;
    private Alquila alquila;

    //1 a 1 con Alquila, relajo de este lado
    public Vehiculo( string color, string matricula, Abonado abonado)
    {
    }
}

```