

EJEMPLO APLICACIÓN APHORA

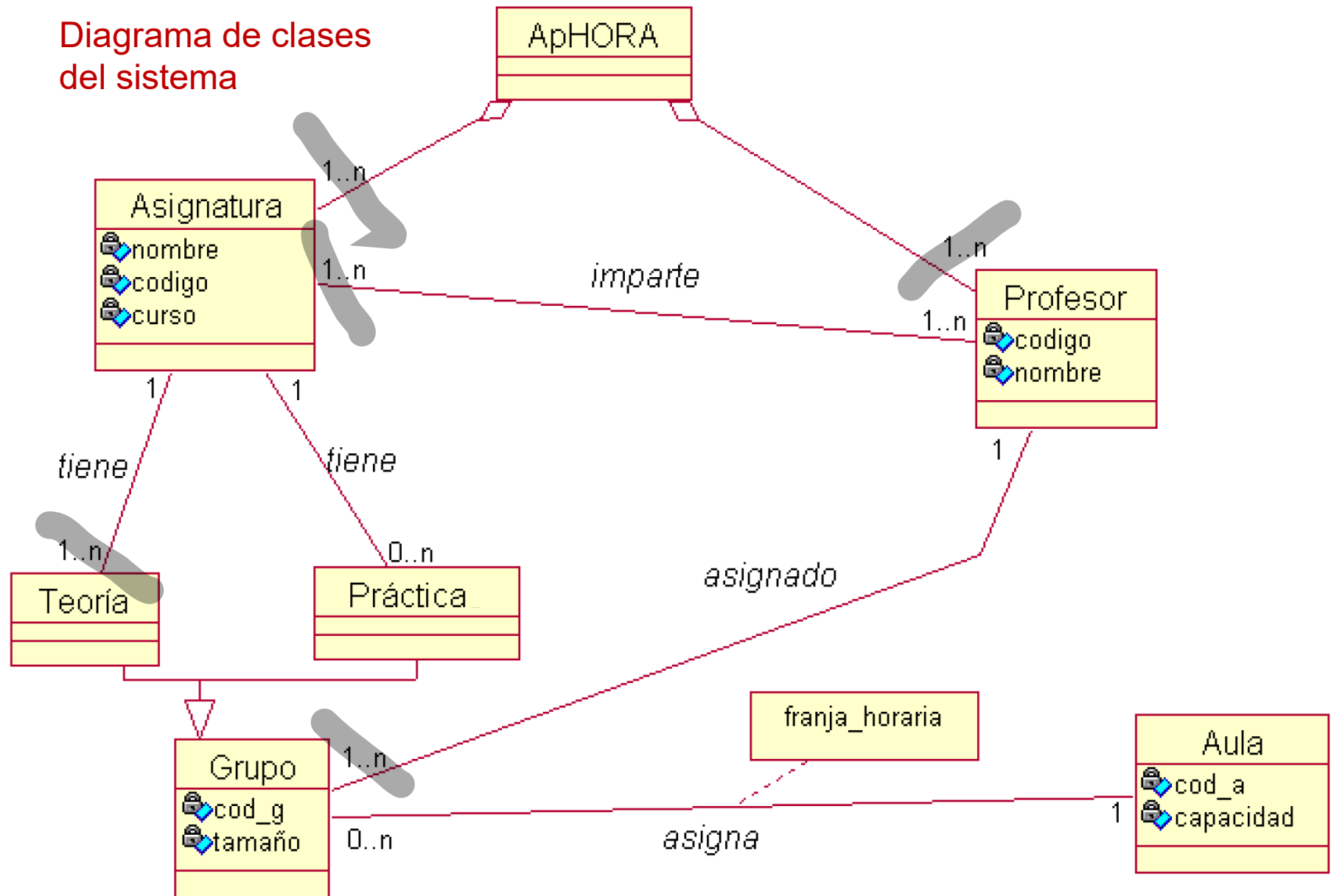
Diseño de clases y constructores
Solución parcial

Ingeniería del Software
ETS Ingeniería Informática
DSIC – UPV

Contenido

- Diagrama de clases del ejemplo
- Diseño en c# de las clases (también sus constructores):
 - ApHora
 - Asignatura
 - Profesor
 - Grupo
 - Teoría

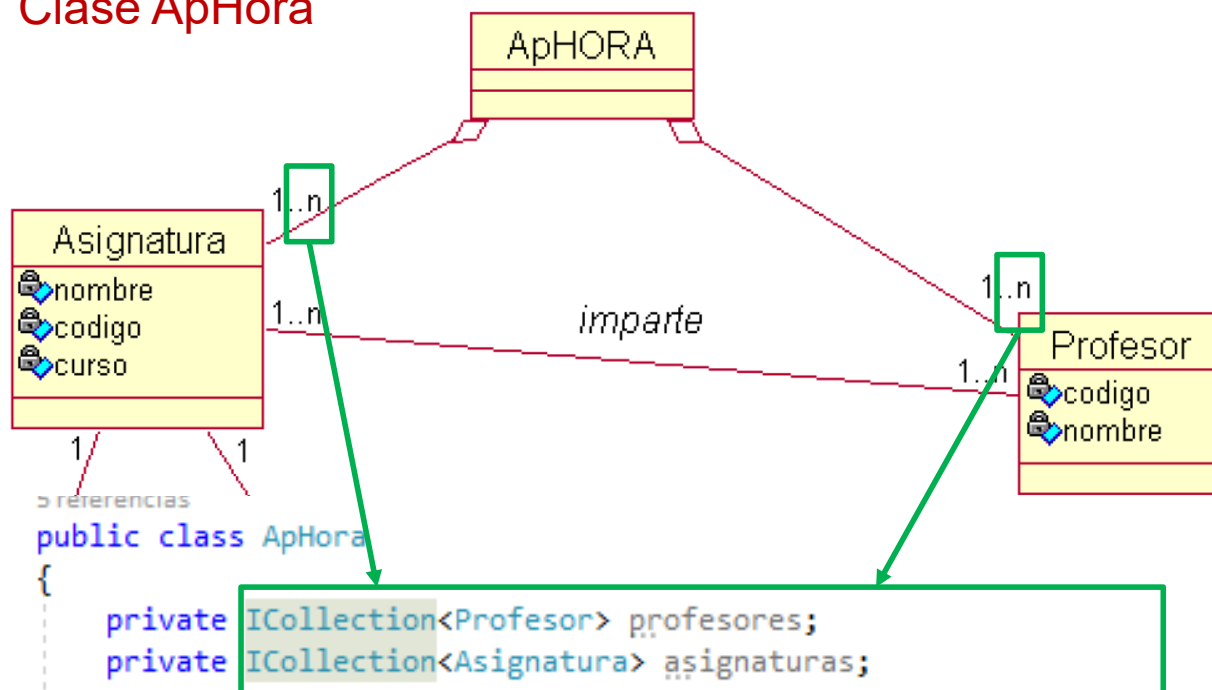
Diagrama de clases del sistema



DISEÑO DE LOS ATRIBUTOS

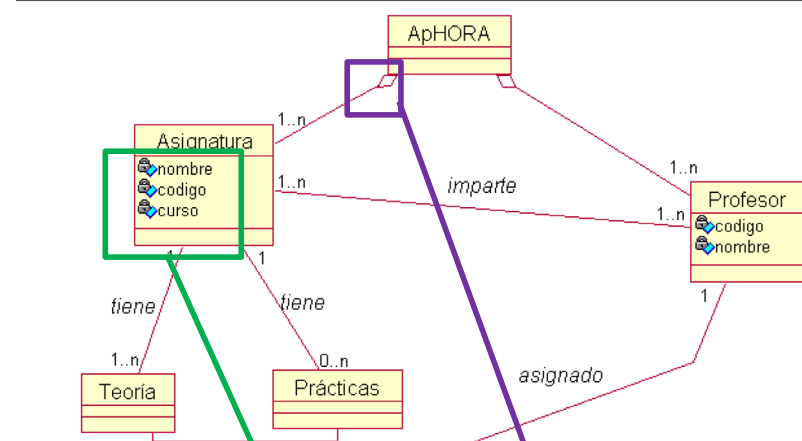
Cardinalidad Máxima

Clase ApHora



La multiplicidad máxima n nos indica que tenemos que crear dos colecciones en ApHORA

Clase Asignatura



Añadimos los atributos propios

La multiplicidad **máxima de 1** nos indica que hay que añadir un atributo de esa clase

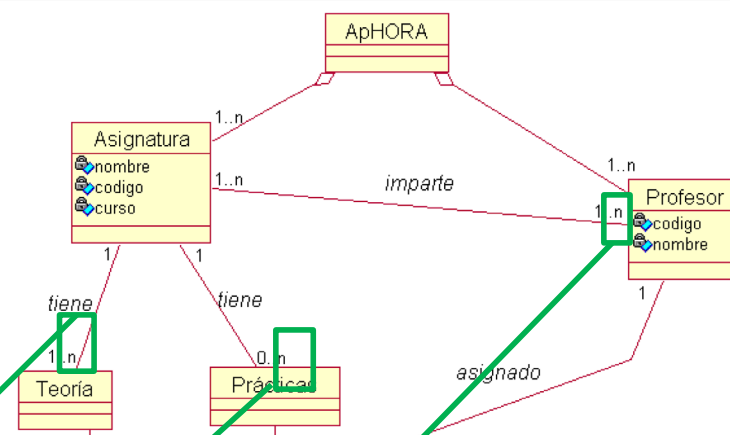
10 referencias

```
public class Asignatura
{
    private string nombre;
    private int codigo;
    private string curso;
    private ApHORA enApHORA;

    private ICollection<Profesor> profesores;
    private ICollection<Teoría> gruposTeoria;
    private ICollection<Prácticas> gruposPrácticas;
```

Clase Asignatura

La multiplicidad máxima n nos indica que tenemos que crear tres colecciones en Asignatura



10 referencias

```
public class Asignatura
```

```
{
```

```
    private string nombre;
```

```
    private int codigo;
```

```
    private string curso;
```

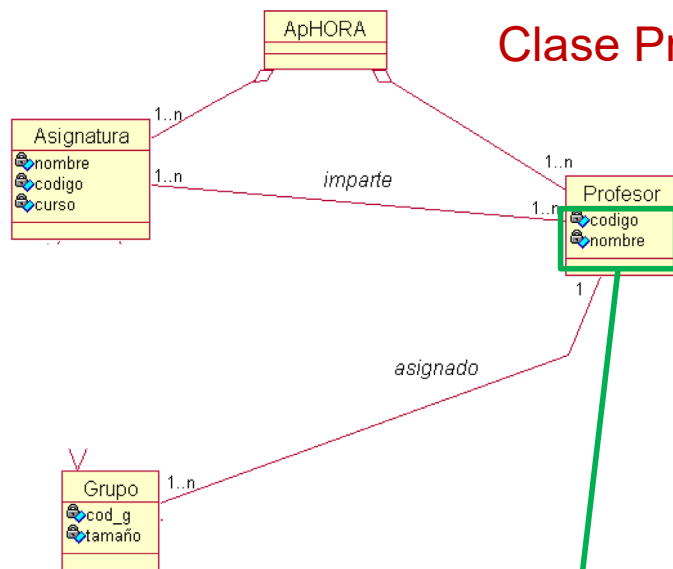
```
    private ApHORA enApHORA;
```

```
    private ICollection<Profesor> profesores;
```

```
    private ICollection<Teoria> gruposTeoria;
```

```
    private ICollection<Practicas> gruposPracticas;
```

Clase Profesor

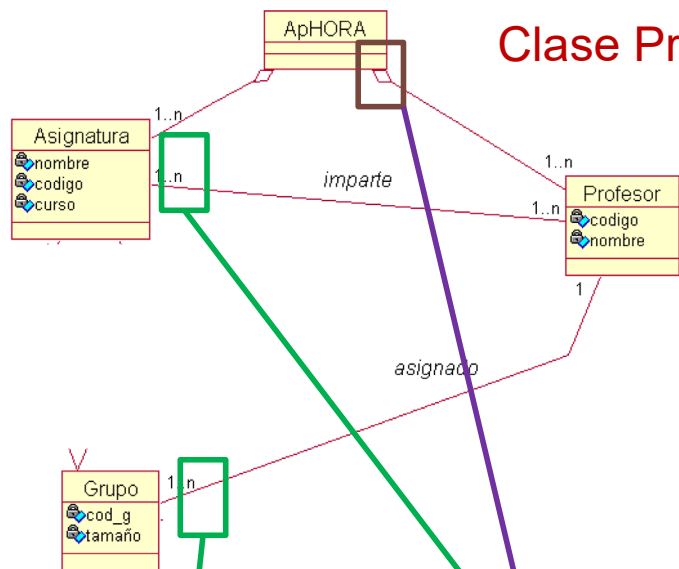


Tenemos que declarar los atributos propios

9 referencias

```
public class Profesor
{
    private int codigo;
    private string nombre;
    private ApHORA enApHORA;
    private ICollection<Asignatura> asignaturas;
    private ICollection<Grupo> grupos;
```


Clase Profesor



La multiplicidad **máxima n** nos indica que tenemos que crear dos colecciones en Profesor

La multiplicidad **máxima de 1** nos indica que hay que añadir un atributo de esa clase

Referencias

```
public class Profesor
```

```
{
```

```
    private int codigo;
```

```
    private string nombre;
```

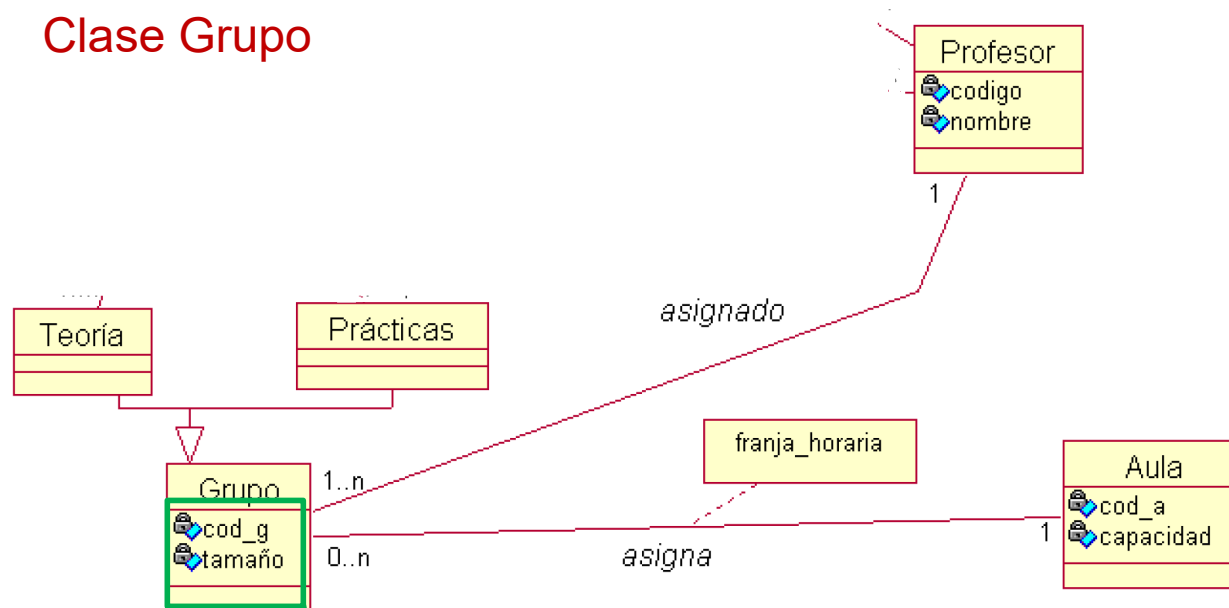
```
    private ApHora enApHora;
```

```
    private ICollection<Asignatura> asignaturas;
```

```
    private ICollection<Grupo> grupos;
```

Clase Grupo

Tenemos que declarar sus atributos



7 referencias

```
public class Grupo
```

```
{
```

```
private int cod_g;
```

```
private int tamaño;
```

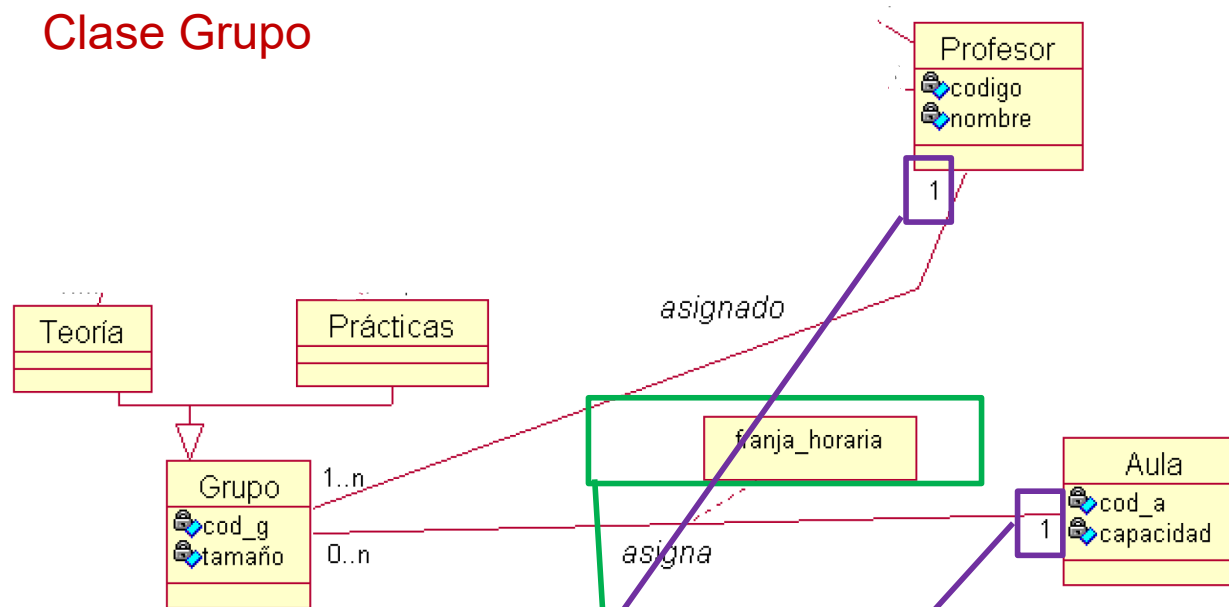
```
private DateTime hora_desde;
```

```
private DateTime hora_hasta;
```

```
private Aula aula;
```

```
private Profesor profesor;
```

Clase Grupo



Atributo de una relación uno a muchos, va al extremo muchos

La multiplicidad **máxima de 1** nos indica que hay que añadir un atributo de esa clase

7 referencias

```
public class Grupo
```

```
{
```

```
    private int cod_g;
```

```
    private int tamaño;
```

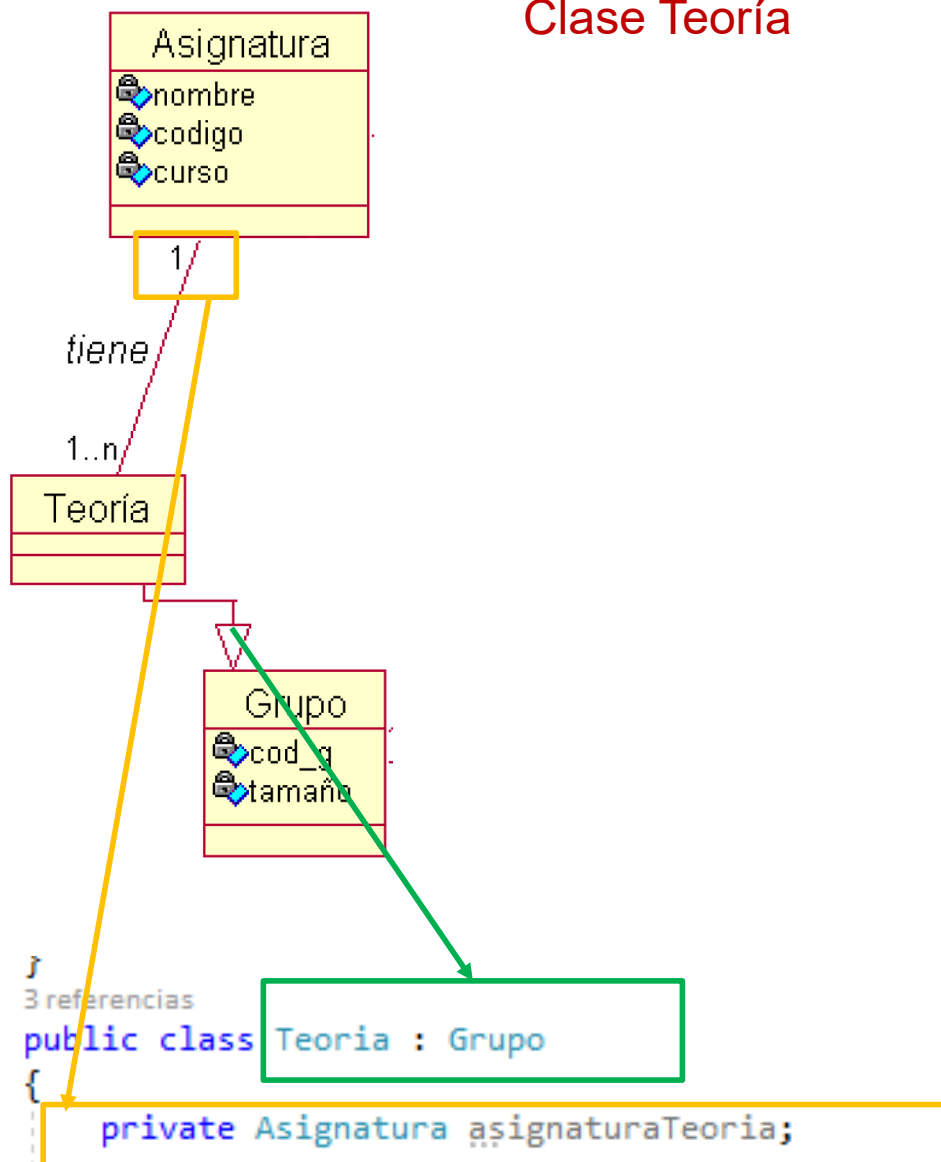
```
    private DateTime hora_desde;
```

```
    private DateTime hora_hasta;
```

```
    private Aula aula;
```

```
    private Profesor profesor;
```

Clase Teoría

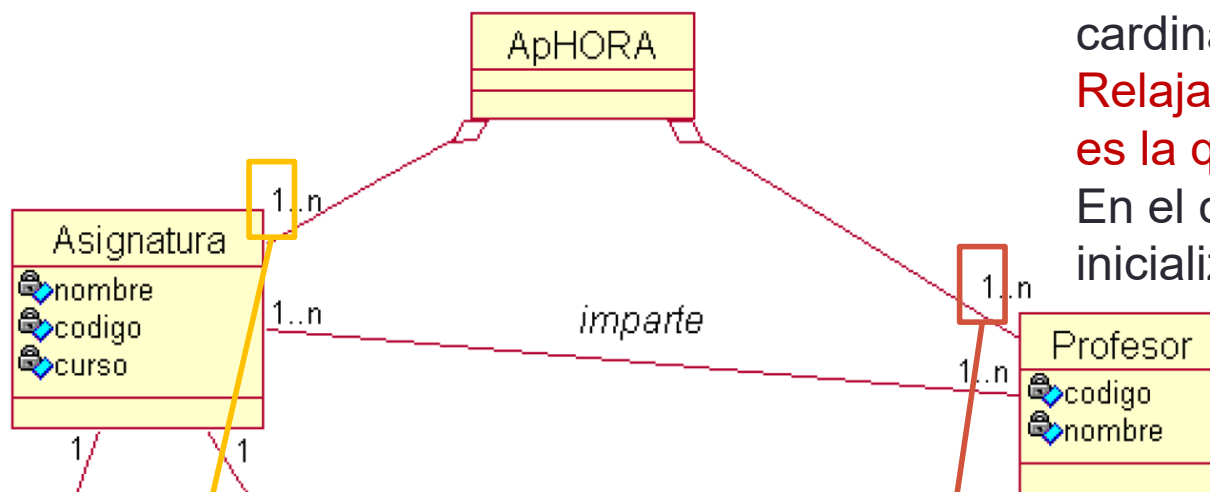


Teoría hereda de grupo

La cardinalidad máxima de 1 nos indica que debemos añadir un atributo de esa clase

DISEÑO DE CONSTRUCTORES

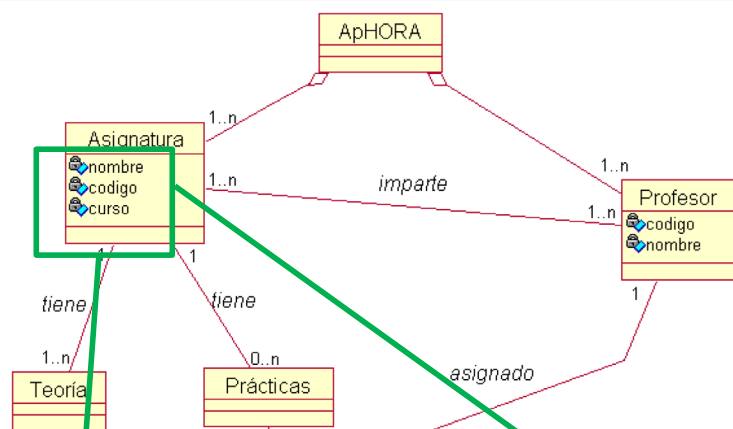
Cardinalidad Mínima



Tenemos las dos relaciones con cardinalidad **mínima 1 a 1**
Relajaremos en esta clase, que es la que tiene las colecciones
En el constructor, solo inicializamos las colecciones

```
public 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  
}
```

Clase Asignatura



El constructor debe recibir como parámetro los valores necesarios para inicializar sus atributos

```

//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 Teoría, relajamos en este lado
// y eliminamos teoria del constructor
1 referencia

```

```

public Asignatura(String nombre, int codigo, String curso, ApHora apHora)

```

```

{
    this.nombre = nombre;
    this.codigo = codigo;
    this.curso = curso;

```

```

    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 codigo

```

```

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

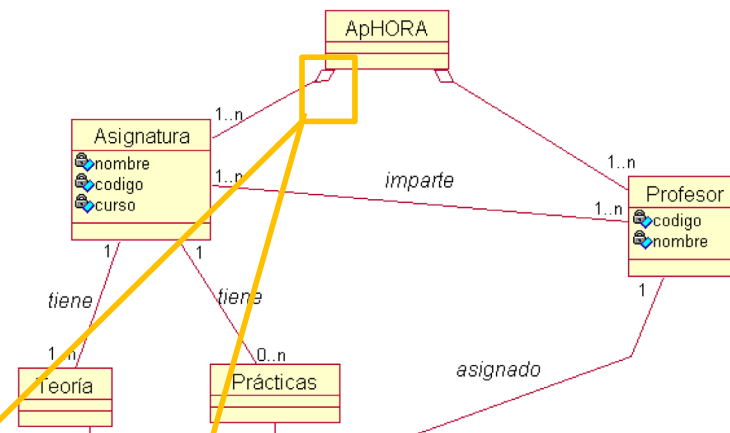
```

```

}

```

Clase Asignatura



```

//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

```

```

1referencia
public Asignatura(String nombre, int codigo, String curso, ApHora apHora)
{
    this.nombre = nombre;
    this.codigo = codigo;
    this.curso = curso;

    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 codigo

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

```

Tenemos una cardinalidad **mínima de 1 con ApHora**, pasamos e instanciamos con el objeto el atributo en el constructor

Clase Asignatura

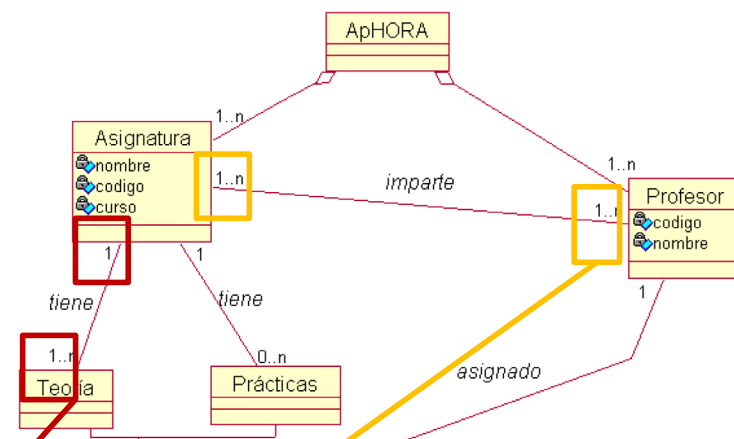
```
//Tenemos una cardinalidad mínima 1 a 1 con profesor, relajamos en es:
// y eliminamos profesor del constructor
//Tenemos una cardinalidad mínima 1 a 1 con ApHora, relajaremos en Apl
//Tenemos una cardinalidad mínima 1 a 1 con Teoria, relajamos en este lado
// y eliminamos teoria del constructor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

    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

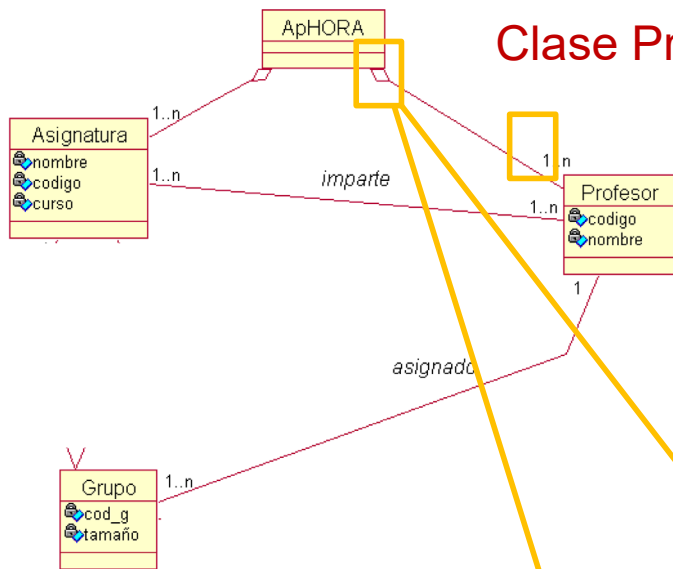
    this.gruposPracticas = new List<Practicas>();
}
```



Tenemos relaciones con cardinalidad **mínima 1 a 1**.
En las colecciones, relajamos en este lado. No pasamos el objeto pero inicializamos las colecciones.

Las restricciones mínimas que relajamos habrá que asegurarlas por código

Clase Profesor



Tenemos las tres relaciones con cardinalidad **mínima 1 a 1**.

Para la relación con ApHora pasaremos **un objeto de la clase en el constructor**, inicializando el atributo con él.

Relajamos en ApHora

```

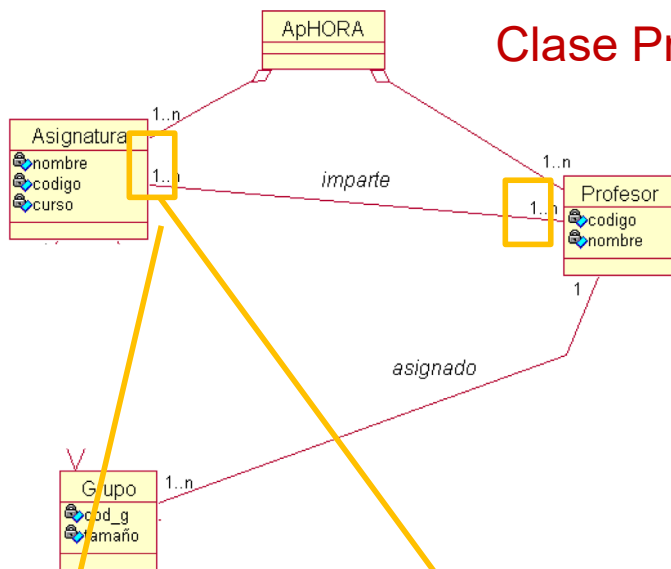
//Tenemos una cardinalidad mínima 1 a 1 con Grupo, relajamos de este lado y eliminamos grupo del constructor
//Tenemos una cardinalidad mínima 1 a 1 con ApHora, relajaremos en ApHora
//Tenemos una cardinalidad mínima 1 a 1 con Asignatura, relajaremos en Asignatura
1 referencia
public Profesor(int codigo, 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
}
  
```

Clase Profesor



Tenemos las tres relaciones con cardinalidad **mínima 1 a 1**.

Para la relación con Asignatura, pasaremos **un objeto de la clase en el constructor**, añadiéndolo a la colección, que se tiene que inicializar antes.

Relajamos en Asignatura

```
//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
```

```
1 referencia
public Profesor(int codigo, string nombre, Asignatura asignatura, ApHora apHora)
```

```
{
    this.codigo = codigo;
    this.nombre = nombre;

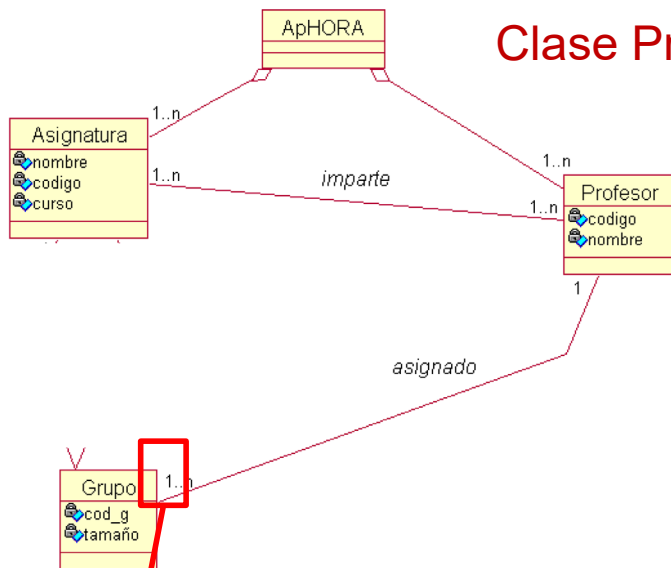
    this.enApHora = apHora;

    this.asignaturas = new List<Asignatura>();
    this.asignaturas.Add(asignatura);
```

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

```
}
```

Clase Profesor



En la relación **mínima 1 a 1 con Grupo**, relajamos en este lado.
En el constructor solo tenemos que inicializar la colección

Habría que asegurar la cardinalidad mínima por código

```

private int codigo;
private string nombre;
private ApHORA enApHORA;
private ICollection <Asignatura> asignaturas;
private ICollection <Grupo> grupos;

```

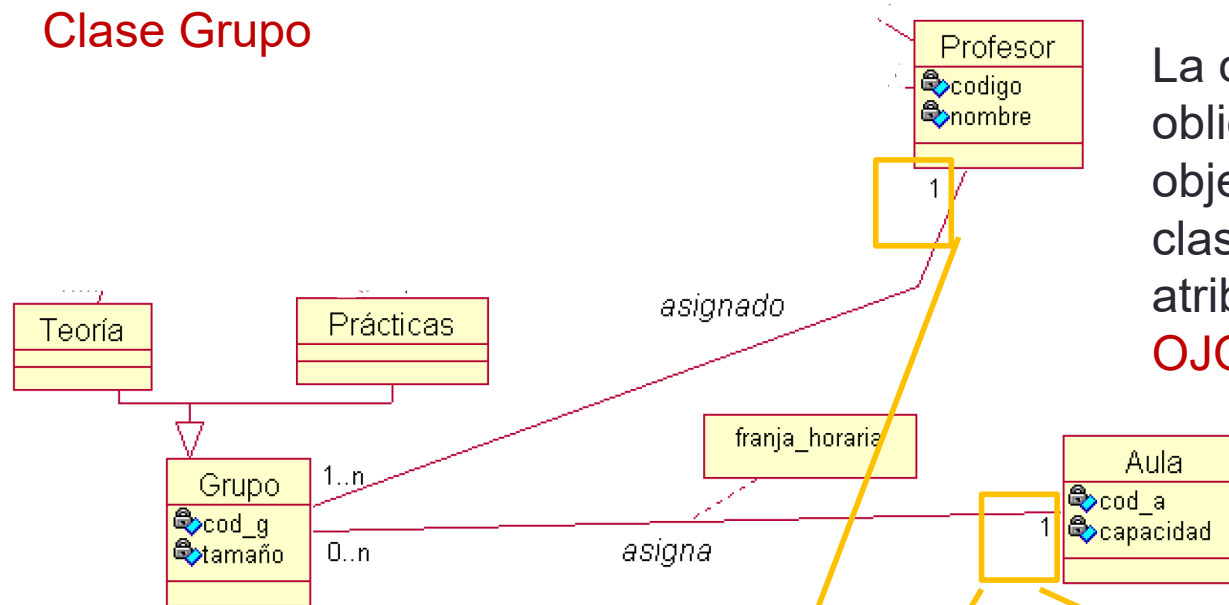
//Tenemos una cardinalidad mínima 1 a 1 con Grupo, relajamos de este lado y eliminamos grupo del constructor
 //Tenemos una cardinalidad mínima 1 a 1 con ApHORA, relajaremos en ApHORA
 //Tenemos una cardinalidad mínima 1 a 1 con Asignatura, relajaremos en Asignatura

```

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
}

```

Clase Grupo



La cardinalidad mínima de 1 nos obliga a pasar al constructor un objeto de la correspondiente clase, así como añadirlo al atributo correspondiente.

OJO con mínimas 1 a 1

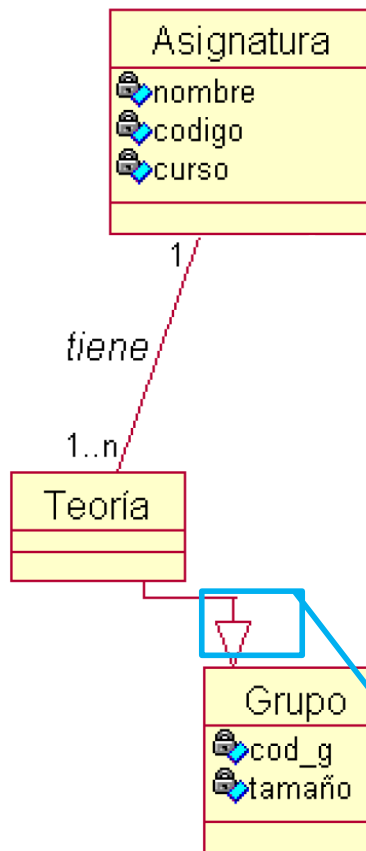
```

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

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

Yellow arrows point from the '1' cardinalities in the UML diagram to the corresponding parameters in the constructor: from 'Profesor' to 'profesor', from 'Aula' to 'aula', and from 'franja_horaria' to the 'asigna' relationship parameters.

Clase Teoría



Teoría es una especialización de Grupo (herencia).

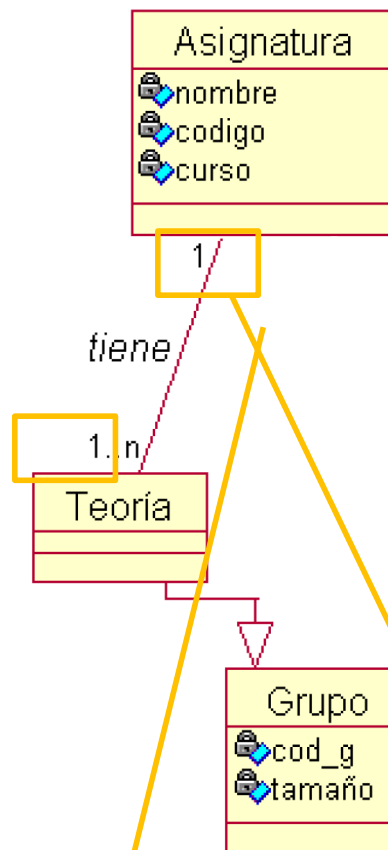
Posee todos los atributos del padre, y hay que pasarlos junto a los propios al constructor e instanciarlos

`:base` -> nos permite llamar al constructor de la clase de la que heredamos, el constructor de Grupo (C#) al que pasamos los valores

Referencias

```
public Teoría(int cod_g, int tamaño, DateTime hora_desde, DateTime hora_hasta, Aula aula,
    Profesor profesor, Asignatura asignatura) : base(cod_g, tamaño, hora_desde, hora_hasta, aula, profesor)
{
    asignaturaTeoría = asignatura;
}
```

Clase Teoría



Tenemos una cardinalidad **mínima de 1 a 1** con Asignatura.

Pasamos un objeto de Asignatura como parámetro en el constructor e instanciamos el atributo correspondiente

Relajaremos en la clase asignatura

```
public class Teoría : Grupo {
    private Asignatura asignaturaTeoría;

    public Teoría (int cod_g, int tamaño, DateTime hora_desde,
        DateTime hora_hasta, Aula aula, Asignatura asignatura): base (cod_g, tamaño, hora_desde, hora_hasta, aula){
        asignaturaTeoría = asignatura;
    }
}
```

A yellow box highlights the line `asignaturaTeoría = asignatura;` in the constructor of the `Teoría` class.

INSTANCIACIÓN DEL SISTEMA

Añadimos las instrucciones para cumplir las restricciones por código

0referencias

class Program

```
{  
    0referencias  
    static void Main(string[] args)  
    {  
        ApHora miApHora = new ApHora();  
  
        Aula aula104 = new Aula(104,70);  
  
        Asignatura isw = new Asignatura("ISW", 1, "tercero", miApHora);  
        Profesor soledad = new Profesor(1, "soledad", isw, miApHora);  
        isw.AddProfesores(soledad); //aseguramos por código la relajación en Asignatura con Profesor  
        Teoria teo_isw = new Teoria(11, 50, new DateTime(2020, 9, 14, 15, 0, 0), new DateTime(2020, 9, 14, 16, 30, 0), aula104,soledad, isw);  
        isw.AddTeoria(teo_isw); //aseguramos por código la relajación en Asignatura con teoria  
  
        soledad.AddGrupos(teo_isw); //aseguramos por código la relajación en Profesor con Grupo  
  
        miApHora.AddAsignaturas(isw); //aseguramos por código la relajación de ApHora con Asignarua  
        miApHora.AddProfesores(soledad); //aseguramos por código la relajación de ApHora con Profesor  
  
        aula104.AddGrupo(teo_isw); //dejamos consistente el modelo|  
    }  
}
```