UNIVERSIDAD POLITECNICA DE VALENCIA

Pràctiques

Butlletí Pràctica 3

Generació Automàtica de Codi i Constructors

Enginyeria del Programari

ETS Enginyeria Informàtica
DSIC – UPV

Curs 2017-2018

1. Objectiu

Generar automàticament codi a partir del model de disseny subministrat i implementar els constructors de cada classe.

2. Passos Inicials

Per a treballar en aquest laboratori **un membre de l'equip** realitzarà els següents passos:

- Obriga l'eina de desenvolupament Visual Studio 2015 Enterprise (Inici > Tots els programes > Microsoft Visual Studio 2015)
- Inicie sessió amb el seu compte de Microsoft prement a "Iniciar sessió" (a dalt a la dreta)
- Connecte's al seu projecte d'equip definit en la sessió 1 (botó amb forma d'endoll en Visual Studio>Team Services)
- Obtinga l'última versió de la seua solució del repositori de Team Services

Els següents passos els realitzarà només un membre de l'equip mentre no s'indique el contrari

3. Configuració inicial de la solució

Com ja s'ha estudiat en les classes de teoria, l'aplicació a desenvolupar tindrà tres capes: **Presentació, Lògica de negoci** i **Persistència**. En primer lloc, anem a estructurar la solució Visual Studio perquè tinga aquestes tres capes. Per a açò, i com ja s'ha estudiat en els seminaris de teoria, la solució de Visual Studio ha de contenir les següents "Carpetes de Solució": Testing, Presentation, Library. Si no les ha creat ja, pot fer-ho de la següent forma: `Botón derecho ratón en la Solución > Agregar > Nueva Carpeta de Soluciones`

Una vegada fet açò la seua solució tindrà, a més de l us projectes de Modelatge de la sessió 2, i el projecte de testing de la sessió 1, l'estructura de la figura 1 (el nom de la solució pot ser diferent al de la figura):

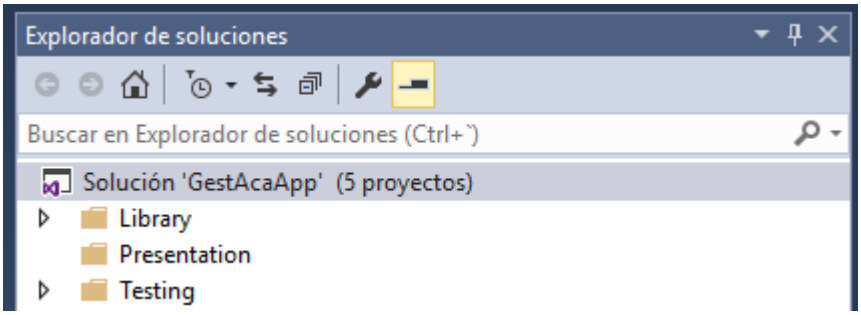

Figura 1. Estructura inicial de la solució

4. Creació d'un projecte de biblioteca de classes

IMPORTANT: No canvie el nom al projecte de biblioteca que es descriu en aquest document i que es denomina **GestDepLib.**

Les capes de persistència i de lògica de negoci van a estar implementades com una biblioteca de classes *GestDepLib* (una dll quan es compile la solució). Per a açò ha d'agregar un projecte de biblioteca de classes a la carpeta **"Library"** de la següent forma: `Botón derecho ratón sobre carpeta Library > Agregar > Nuevo proyecto > VisualC# > Windows > Biblioteca de Clases > GestDepLib`

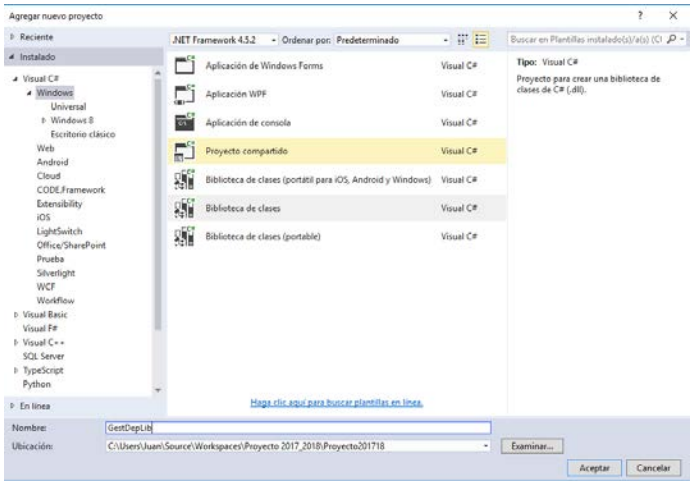

Figura 2. Finestra per afegir un nou projecte de biblioteca de classes

Una vegada creat el projecte de biblioteca de classes li agregarem dues carpetes (*BusinessLogic* i *Persistence*). Per a açò realitzarà dues vegades la següent operació: `Botón derecho de ratón sobre proyecto GestDepLib > Agregar > Nueva Carpeta.` A més, afegisca una subcarpeta `Entities` en `BusinessLogic` i una altra carpeta també denominada `Entities` en `Persistence.` Una vegada realitzat açò, la seua solució serà com la de la figura 2:

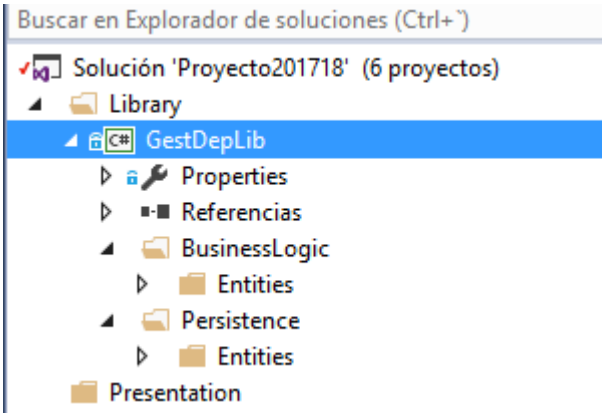

Figura 3. Solució amb projecte de biblioteca de classes creat

5. Important projecte de modelatge de disseny

En la sessió de laboratori 2, l'equip va crear uns projectes de modelatge que no es corresponen amb el cas d'estudi que treballarem a partir d'aquesta sessió. A partir d'aquesta sessió **TOTS** els grups de pràctiques anem a treballar amb **el mateix model** (una versió simplificada del model solució). El diagrama de classes de disseny sobre el qual anem a treballar i que prendrem com a base per a la implementació és el mostrat en la següent figura.

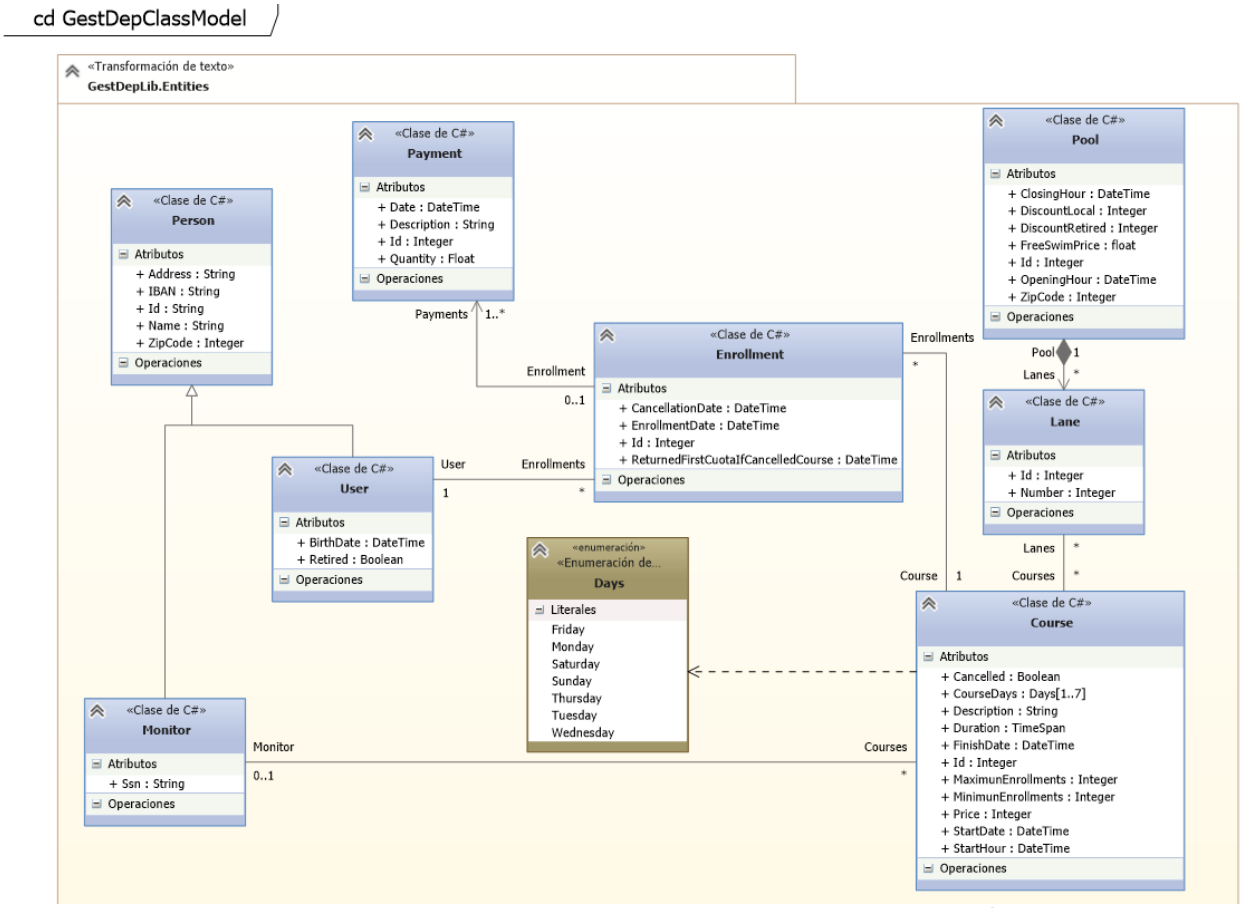

Figura 4. Diagrama de classes de disseny per a la implementació

Per a incorporar aquest model a la seua solució:

- Descarregue de PoliformaT el fitxer comprimit de modelatge que hi ha en la carpeta d'aquesta sessió de laboratori.
- Descomprimisca el fitxer en la seua àrea de treball local (workspace de la solució en el disc C:\)
- Torne a Visual Studio i faça `Botó dret del ratolí sobre la solució principal > Agregar > Proyecto Existente… >Seleccionar el projecte` **`GestDepModel.modelproj`** `desde l'ubicació en disc on estiga descomprimit`

En fer açò la seua solució tindrà el següent aspecte:

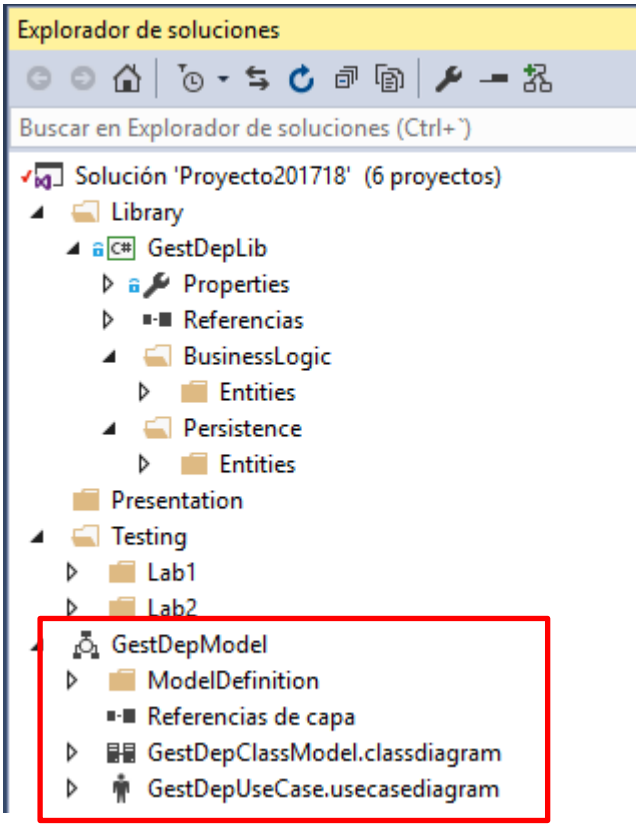

Figura 5. Solució amb projecte de modelatge GesDepModel

6. Generació automàtica de codi

Si obri el diagrama **`GestDepClassModel.classdiagram`** observarà que conté un diagrama com el de la Figura 4. Diagrama de classes de disseny per a la implementacióFigura 4. En aquest diagrama es pot observar que alguns atributs porten afegit un "?" al costat del tipus de dada. En la classe **Enrollment** podem veure els següents atributs amb aquesta característica:

```
CancellationDate: DateTime?
ReturnedFirstCuotaIfCancelledCourse: DateTime?
```

Això indica que el valor del atribut amb un tipus de dada que no es un objecte, pot admetre valors nuls. Las referències a objectes poden ser nul·les, però no els valors (tipus primitius i struct, com ara el tipus DataTime). Aquesta notació permet de fer-los nuls també[1]. Per a comprovar si un atribut d'aquests conté el valor `null` es fa servir la propietat `HasValue`, i per obtindré el valor la propietat `Value`. Al nostre exemple, la data de cancel·lació (`CancelationDate`) hi podria mancar, per això hem de permetre que agafi valor nul, i per la definim com de tipus `DateTime?`

A partir d'aquest diagrama de disseny, Visual Studio és capaç de generar automàticament el codi associat en llenguatge de programació C# (classes, atributs, atributs que implementen associacions). Per a generar el codi: `Obrir el diagrama de classes de disseny (Design.classdiagram) > Polse botó dret ratolí sobre el paquet denominat GestDepLib.Entities > Generar Código`

[1] https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/nullable-types/using-nullable-types

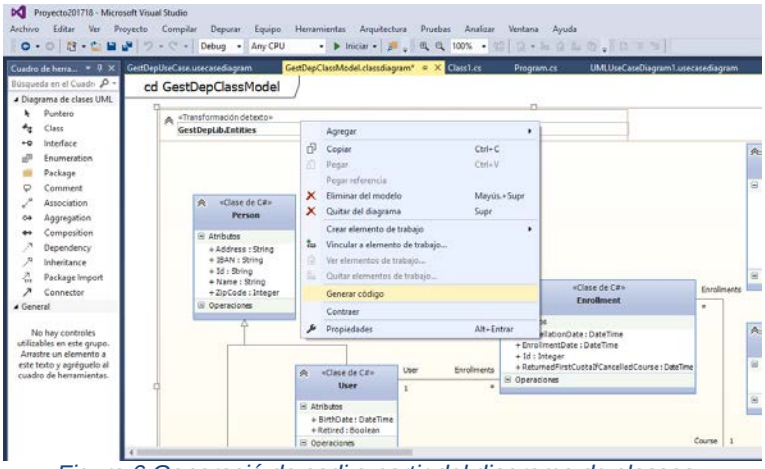

Figura 6 Generació de codi a partir del diagrama de classes

Al finalitzar el procés de generació de codi s'haurà generat un projecte denominat `GestAcaModelLib` com el de la Figura 7 i que conté una carpeta `GeneratedCode`. Aquesta carpeta conté totes les classes C# que han sigut generades automàticament per Visual Studio a partir del seu model de disseny.

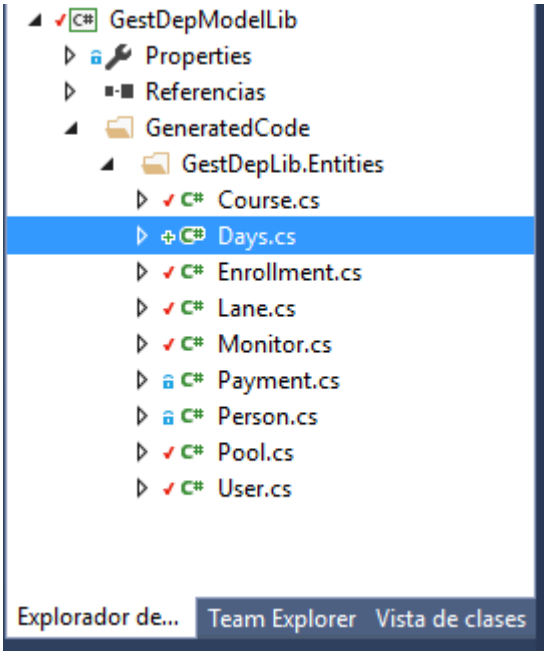

Figura 7. Solució amb codi generat automàticament

Observe les classes generades i veja com les associacions han sigut convertides en atributs seguint les regles de disseny vistes en classe de teoria. Per exemple, l'agregació entre les classes `Course` i `TaughtCourse` del diagrama s'ha convertit en una propietat de tipus `ICollection<TaughtCourse>` en la classe `Course` i en una propietat de tipus `Course` en la classe `TaughtCourse`. Tot açò AUTOMÀTICAMENT[2].

La següent figura mostra un fragment de la classe parcial *Enrollment*:

[2] Veja la sección 9 del document per a la plantilla de generació de codi

```
public partial class Enrollment
{
    public DateTime EnrollmentDate
    {
        get;
        set;
    }

    public DateTime? CancellationDate
    {
        get;
        set;
    }

    public int Id
    {
        get;
        set;
    }

    public DateTime? ReturnedFirstCuotaIfCancelledCourse
    {
        get;
        set;
    }
```

Figura 8. Fragment de la classe generada Enrollment ambs dos atributs "nullable"

7. Incorporació classes generades a biblioteca de classes

A continuació, incorporarem totes aquestes noves classes a la nostra biblioteca de classes creada en el punt 3 `GestDepLib`. Per a açò realitze els següents passos: Seleccione **en *Visual Studio*** tots els fitxers existents en `GeneratedCode/GestDepLib.Entities` i arrossegue'ls (còpia) **sense eixir de Visual Studio** fins a la carpeta `Persistence/Entities` del projecte `GestDepLib.` La solució ha de quedar com la de la Figura 9.

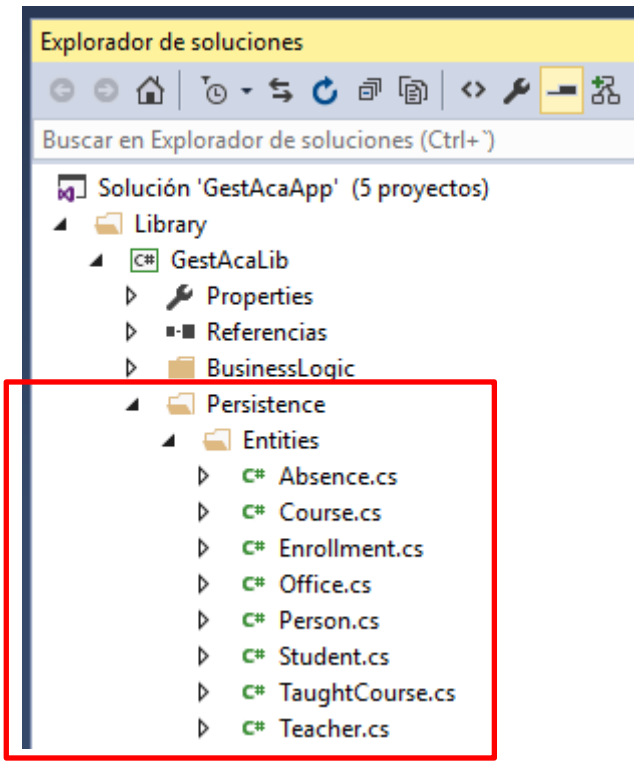

Figura 9. Classes generades automàticament en carpeta Persistence>Entities

Les classes que han sigut generades automàticament són classes parcials (**partial**). Açò vol dir que són classes on la implementació de les quals pot estar fragmentada/dividida en més d'un fitxer. Usarem classes parcials per a separar l'aspecte de persistència de l'aspecte de lògica de negoci per a cada classe del nostre model. Per tant, anem a generar una classe parcial en la carpeta `BusinessLogic/Entities` per a cada classe que tenim en `Persistence/Entities`. Observe que això **no s'aplica** al tipus enumerat. Per a aconseguir-ho realitzarà tantes vegades com siga necessari: `Botó dret ratolí sobre la carpeta BusinessLogic/Entities > Agregar > Clase.` Açò crearà **classes buides** en `BusinessLogic/Entities i` o projecte quedarà de la següent forma:

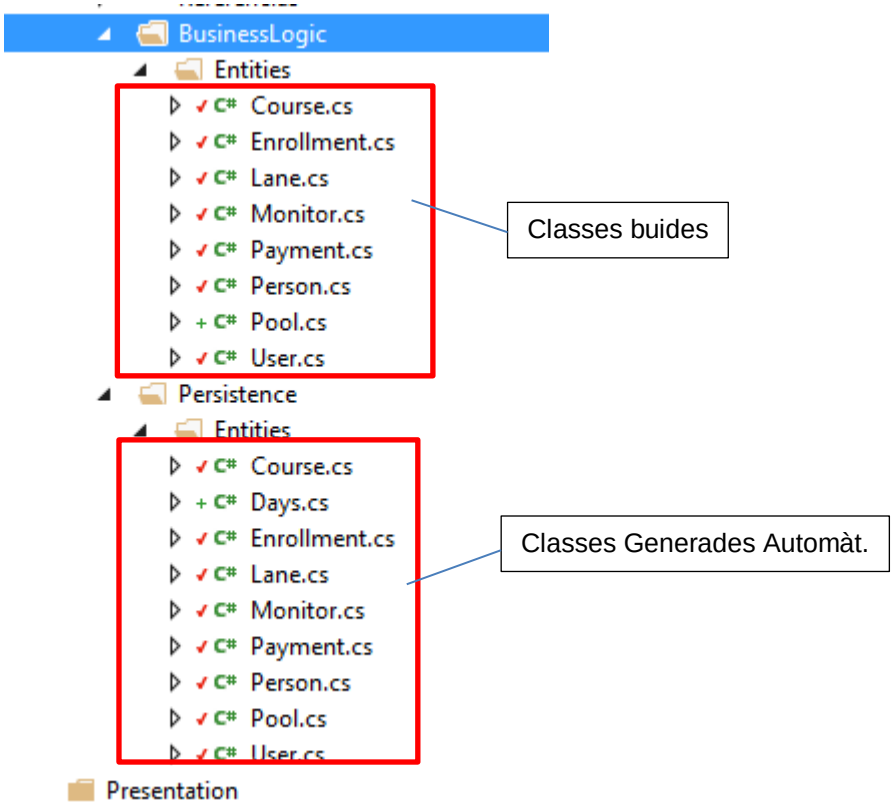

Figura 10. Classes parcials en BusinessLogic/Entities i Persistence/Entities

Ara és moment d'editar **les classes buides de BusinessLogic/Entities**. Per a que aquestes classes siguen també públiques i parcials afegisca `public partial` davant de `class`. Modifique també el namespace de cada classe per a que siga `GestDepLib.Entities`. Per exemple, la classe `Course` de `BusinessLogic/Entities` quedaria de la següent forma:

```
namespace GestDepLib.Entities
{
    using ...

    public partial class Course
    {


    }
}
```

Figura 11. Exemple de classe pública i parcial buida en BusinessLogic/Entities

Protegisca la seua solució per a pujar aquesta versió al servidor. Pose com a comentari "Codi Generat Automàticament".

8. Creació de constructors

A partir d'aquest moment pot dividir el treball entre tots els membres de l'equip. La resta de membres poden obrir Visual Studio, connectar-se al servidor i baixar-se la solució que acabem de protegir en el punt anterior.

A continuació, passarem a definir els constructors de les classes del model. **Els constructors es definiran en les classes parcials situades en BusinessLogic/Entities.**

Tota classe del nostre sistema **tindrà dos constructors**, un sense paràmetres i un altre amb paràmetres. El constructor sense paràmetres simplement inicialitzarà les col·leccions que tinga la classe creant llistes buides. Per exemple, per a la classe `Course`, el seu constructor sense paràmetres seria:

```
public Pool()
{
    Lanes = new List<Lane>();

}
```

En canvi, el segon constructor amb paràmetres per a aquesta mateixa classe seria:

```
public Pool(DateTime OpeningHour, DateTime ClosingHour,
            int ZipCode, int Id, int DiscountRetired, int DiscountLocal,
            float freeSwimPrice)
        {
            this.OpeningHour = OpeningHour;
            this.ClosingHour = ClosingHour;
            this.ZipCode = ZipCode;
            this.Id = Id;
            this.DiscountRetired = DiscountRetired;
            this.DiscountLocal = DiscountLocal;
            this.FreeSwimPrice = freeSwimPrice;
            Lanes = new List<Lane>();
        }
```

Tal com s'ha vist en les classes de teoria, el constructor de la classe `Pool` no té un paràmetre de tipus `TaughtCourse` perquè la cardinalitat mínima de la relació `Pool -> Lane` és zero. Recorde que si entre dues classes A i B la relació de A -> B té cardinalitat mínima 1 llavors el constructor d'A que porta paràmetres ha de tenir un paràmetre de tipus B:

```
public A(/*altres paràmetres*/, B b)
```

Implemente els constructors de totes les classes seguint les regles de constructors descrites amb especial atenció a les cardinalitats mínimes de les relacions del model de disseny de la Figura 4 i **NO OBLIDE INICIALITZAR LES PROPIETATS DE COL·LECCIONS COM A LLISTES BUIDES.**

Dividisca el treball en equip de manera que diferents membres treballen sobre diferents classes per a no generar conflictes

IMPORTANT: Quan haja acabat totes aquestes activitats PROTEGISCA la seua solució (açò pujarà i emmagatzemarà el seu codi en el servidor de Team Services), resolga conflictes si els hi hagués, afegisca un comentari que descriga aquesta versió i prema el botó protegir. Addicionalment, per doble

seguretat faça una còpia de la solució des de la seua workspace local (C:\...) a un altre directori permanent (pendrive, etc). El contingut de C:\ no és permanent al laboratori.

9. Especificar plantillas per a la generació de codi

Tal como s'ha vist al seminario 3 de teoria, existeixen dues vies per a modificar/personalitzar la generació de codi:

- Modificant manualment les propietats de les classes, les relacions entre classes, així como els atributs d'aquests elements
- Modificant la plantilla T4 de generació de codi que inclosa a la instalación de Visual Studio: CsharpHelper.T4

La plantilla es trova dins de:

```
C:\Program Files (x86)\Microsoft VisualStudio
14.0\Common7\IDE\Extensions\Microsoft\ArchitectureTools\Extensibility\Templates\Text
```

Per defecte, els atributs generats per a navegar les relacions amb multiplicitat de destí superior a 1, son de tipus IEnumerable. Als laboratoris s'ha modificat la plantilla per a generar un atribut de tipus ICollection (necessari per emprar Entity Framework). Existeix una copia d'aquesta plantilla a Poliformat, que pot descarregarse si es vol emprar en una instalación personal.