UNIVERSIDAD POLITECNICA DE VALENCIA

Pràctiques

Butlletí Pràctiques 2 i 3

Disseny OO. Capa lògica. Disseny de classes i constructors

Enginyeria del Programari
ETS Enginyeria Informàtica
DSIC – UPV

Curs 2024-2025

1. Objectiu

L'objectiu de les dues pròximes sessions de laboratori és obtenir el codi inicial de la capa lògica de l'aplicació del cas d'estudi. Així, a partir del diagrama de disseny que es proporcionarà, els alumnes hauran d'implementar totes les classes indicades utilitzant el llenguatge C#.

2. Connectar-se al projecte i recuperar el repositori

Cada integrant de l'equip pot connectar-se des de Visual Studio al projecte d'A*zure DevOps,* per a **clonar el repositori remot** al repositori local, en el cas d'iniciar sessió als equips del laboratori. Si estem utilitzant un equip privat en el qual ja es va clonar el repositori prèviament, només serà necessari sincronitzar-los per a obtenir els últims canvis. La Figura 1 mostra els passos a seguir per connectar-se al repositori (en cas de tindre dubtes, podeu consultar el seminari 2 o el butlletí de la pràctica 1).

Els apartats 3 i 4 d'aquest butlletí ha de realitzar-los només un membre de l'equip (per exemple, el responsable o coordinador de l'equip).

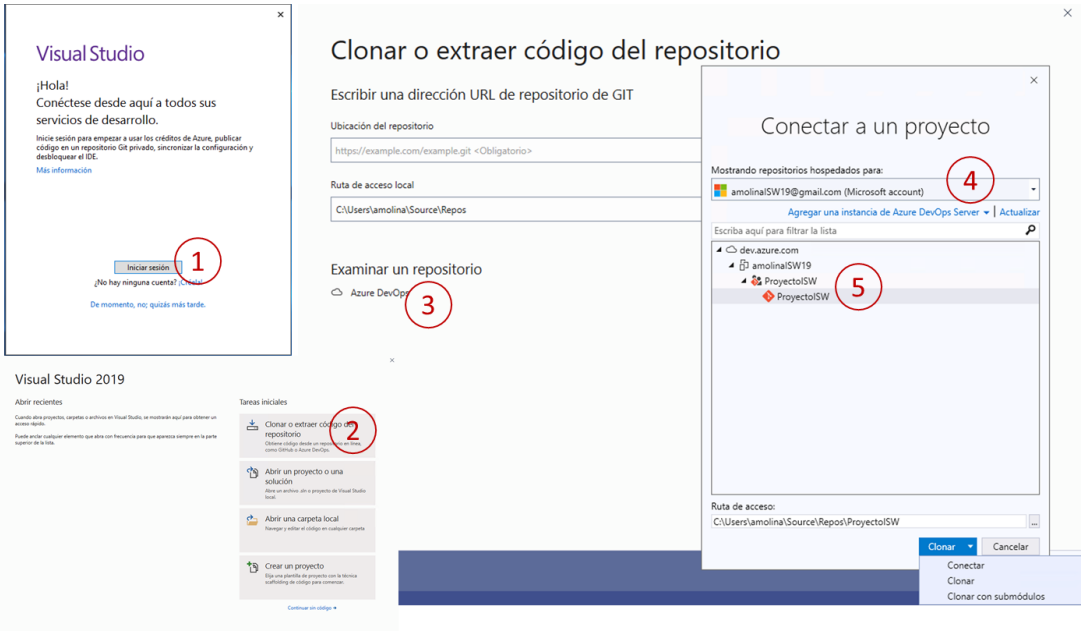

Figura 1. Passos a seguir per connectar-se i clonar el projecte de l'equip.

3. Configuració inicial de la solució

Aquest pas l'ha de fer **únicament el responsable de l'equip,** per tal d'evitar conflictes innecessaris al repositori.

Com ja s'ha estudiat a les classes de teoria i seminari, l'aplicació a desenvolupar va a tindre tres capes: Presentació, Lògica de negoci i Persistència. A les sessions anteriors de pràctiques i seminari ja es va incorporar al projecte una llibreria de classes, que a més a més vàrem preparar amb tres carpetes de solució per a estructurar adequadament el nostre codi: *Library*, *Presentation* i *Testing.* A banda, dins de la carpeta *Library* afegírem dues subcarpetes denominades *BusinessLogic* i *Persistence*.

En aquesta sessió, començarem a afegir codi en aquestes dues carpetes (*BusinessLogic i Persistence*). En primer lloc, comproveu que la vostra solució té el mateix aspecte que l'indicat a la Figura 2. En cas contrari, el responsable de l'equip haurà de reprendre el seminari Visual Studio integrat amb Devops i Git, i el butlletí de la pràctica 1 per tal de completar les passos que falten a fi d'obtenir l'estructura del projecte requerida.

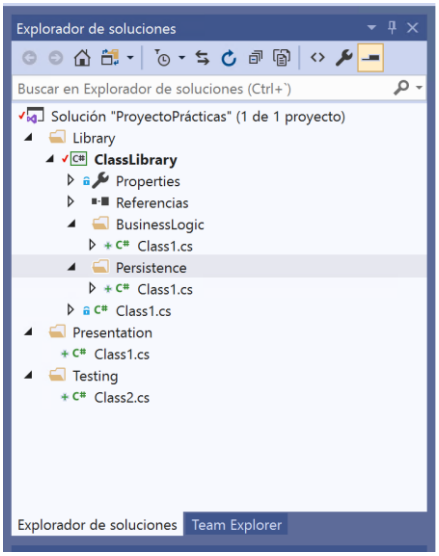

Figura 2. Configuració inicial de la solució del projecte

Després d'obrir la solució, el responsable ha de crear una carpeta de solucions denominada *Entities* dins de la carpeta *BusinessLogic* (mitjançant l'opció `Agregar > Nueva Carpeta` del menú contextual que s'obri al polsar el botó dret del ratolí). A més, afegirem una classe buida per evitar problemes amb el git.

A continuació, s'ha de realitzar el mateix pas per a la carpeta *Persistence,* afegint també una subcarpeta *Entities* i una classe buida dins.

En aquest moment, el responsable ha de realitzar un *commit* amb els canvis, prement el botó `Cambios` de GIT tal i com es mostra a la Figura 3. A més ha d'escriure un missatge de text que descriu el conjunt de canvis realitzats, i polsar el botó `Confirmar todo` del desplegable. Després, ha de pujar els canvis al repositori remot seleccionant l'opció un `Insert` .

La resta de components de l'equip han de sincronitzar els seus repositoris per tal de comprovar que els canvis s'han pujat adequadament amb l'opció `Extraer` . També és possible realitzar les sincronitzacions d'entrada i eixida al mateix temps triant `Confirmar todo y Sincronizar` en el desplegable.

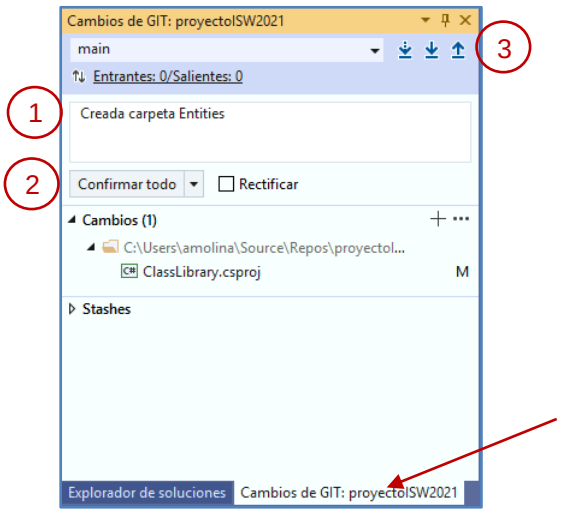

Figura 3. Insertando los cambios en el repositorio (push)

Seguidament, el responsable de l'equip va a canviar el nom i a configurar la llibreria de classes que afegírem a les sessions anteriors. Des de `l'Explorador de Soluciones`, ha de seleccionar la llibreria denominada ClassLibrary i prémer el botó dret del ratolí. Del menú contextual que hi apareix, ha de triar l'opció `Cambiar Nombre`, ficant-li `GestAcaLib` com a nou nom (veure Figura 4).

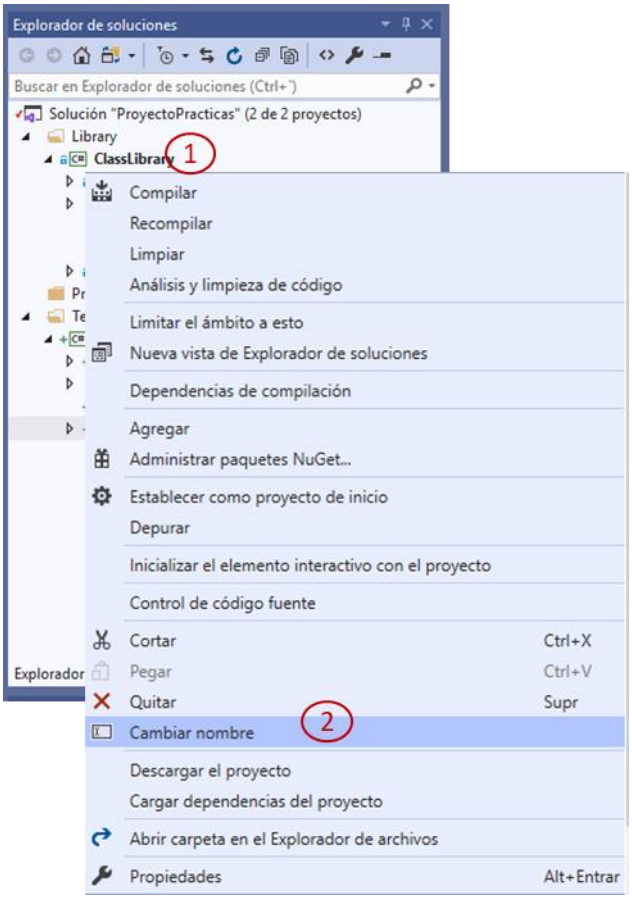

Figura 4. Canviar el nom de la llibreria

A continuació, el responsable ha de configurar el *Namespace* i el nom de la llibreria. Per tant, ha de seleccionar novament la llibreria, prémer el botó dret del ratolí i triar l'opció `Propiedades` del menú contextual. A l'esquerra s'obrirà la fitxa de propietats de la llibreria. Ha de modificar el camp `Nombre del ensamblado` pel valor: `GestAcaLib.` A més, també ha de modificar el nom de l'espai de noms predeterminat pel valor: `GestAca`. La Figura 5 resumeix els passos a seguir.

Per últim, el responsable eliminarà la classe Class1.cs creada per defecte a l'arrel de la llibreria de classes (**IMPORTANT: no són les que estan dins de les carpetes per evitar problemes amb git, és la que està al mateix nivell que les carpetes**).

Arribats a aquest punt, el responsable ha de tornar a *Confirmar* tot el canvis i pujar-los al repositori remot, tal com s'ha explicat anteriorment (Figura 3). Novament, els companys poden recuperar aquestos canvis i comprovar les modificacions.

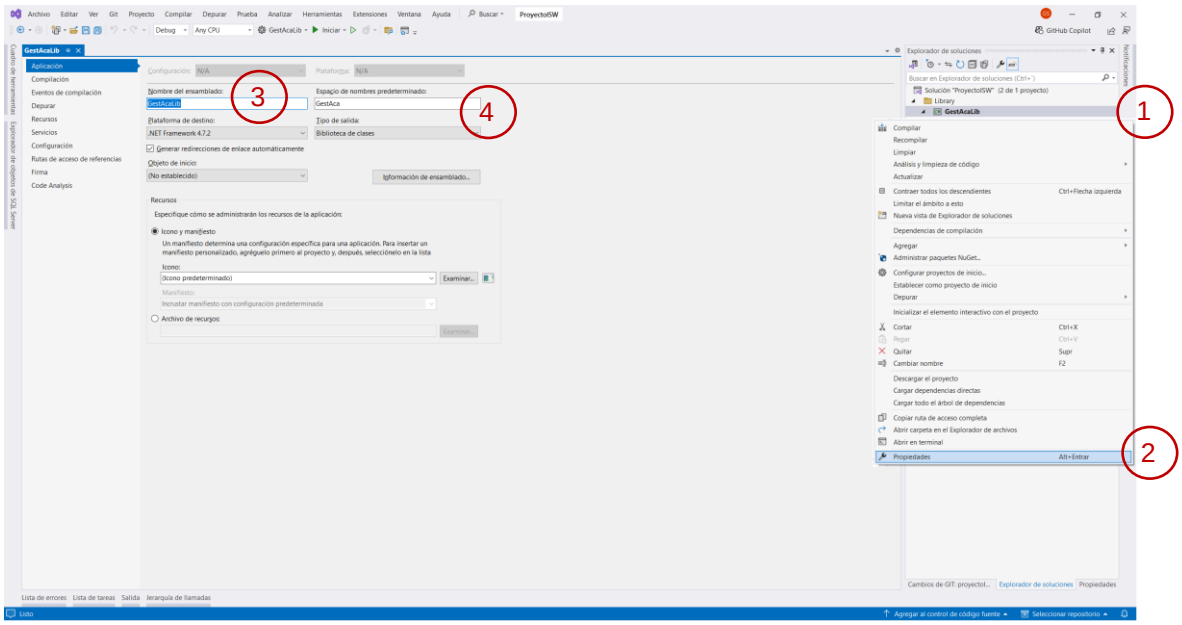

Figura 5. Canviar el nom de l'ensamblat de la llibreria i el nom del espai de noms

4. Afegir les classes del model de disseny

A la Figura 6 teniu el model de disseny de l'aplicació que heu d'implementar. Està formada per 8 classes. La implementació de la vostra solució **ha de respectar el nom de les classes i els atributs**, així com reflectir fidelment les relacions establides al model. Heu de mirar la navegabilitat de les associacions del model, per defecte son bidireccional i es tenen que implementar en el dues sentits de navegació (les unidireccionals sols s'implementen en el sentit indicat per la fletxa, si es que hi han al model).

4.1 Creació de la primera classe

Anem a afegir un fitxer de classe per cadascuna de les classes del model a les carpetes que acabem de crear. Utilitzarem l'estratègia de **classes parcials** que ens permet C# (**`public partial class`**), de forma que la implementació d'una classe pot estar repartida en més d'un fitxer. Utilitzarem classes parcials per poder separar l'aspecte de persistència de l'aspecte de lògica de negoci per a cada classe del nostre model. En concret, separarem la implementació d'una classe en dos arxius, un en la carpeta Persistence/Entities que inclourà la declaració de les propietats de la classe, i un altre en la carpeta BusinessLogic/Entities que contindrà la lògica de la classe (constructor i mètodes).

Novament, els següents passos **només els ha de realitzar el responsable de l'equip** per evitar conflictes innecessaris al repositori. En primer lloc, ha de situar-se dins de la carpeta *Persistence/Entities*.

Els passos per agregar una classe a la carpeta són (veure Figura 7):

1. Situat sobre la carpeta de solucions *Persistence/Entities* i obri el menú contextual amb el botó dret del ratolí.
2. Selecciona `Agregar>Nuevo Elemento`
3. Selecciona l'element `Clase` (es l'element predeterminat)
4. Indica el nom de la classe a crear (Student.cs)
5. Prémer `Agregar`

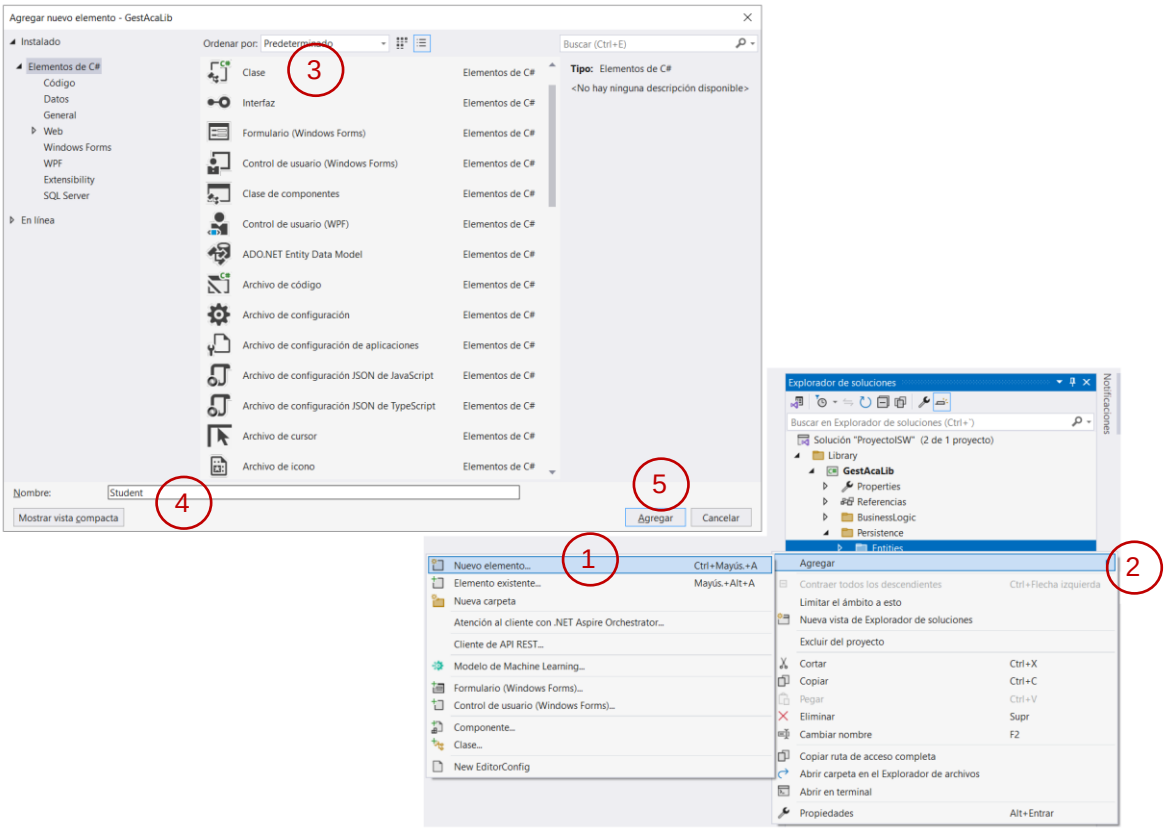

Figura 7. Agregar una nova classe al projecte

En el editor, canviar el *namespace* i la definició de la classe, i el contingut quedarà com el següent codi:

```
namespace GestAca.Entities
{
    public partial class Student
    {
    }
}
```

Torna a guardar els canvis amb un *commit* i puja'ls al repositori remot. Novament, els companys poden recuperar aquests canvis i comprovar les modificacions.

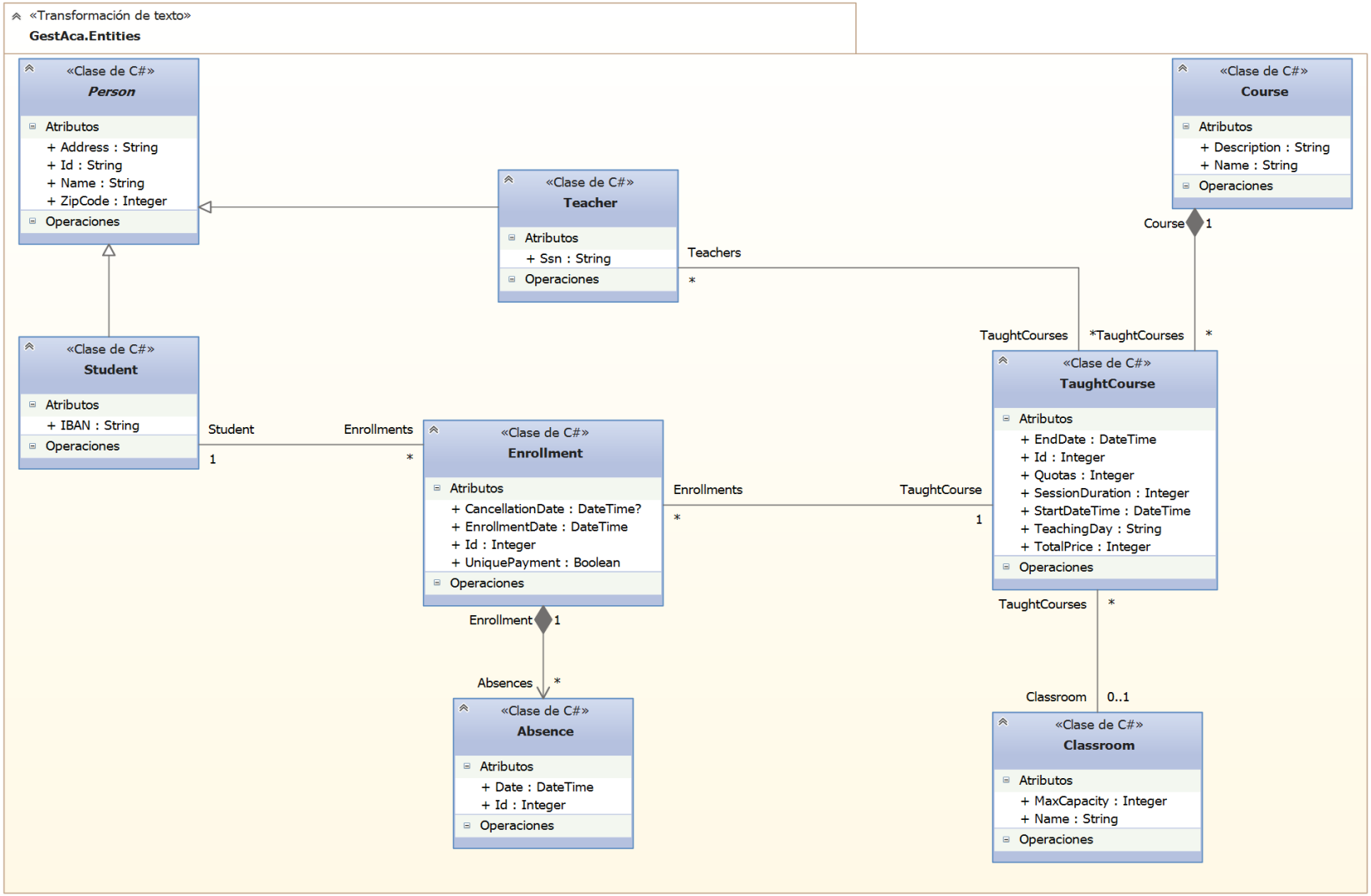

Figura 6. Diagrama de classes de disseny del cas d'estudi

4.2 Creació de la plantilla de classes

Abans de continuar, anem a utilitzar una plantilla per a crear totes les classes com a públiques i parcials dins del mateix *namespace.* Per tant, el responsable de l'equip:

- Selecciona del menú principal l'opció `Projecte > Exportar Plantilla`.
- Apareixerà un diàleg per a triar el tipus de plantilla. Selecciona l'opció `Plantilla de elemento` i prem `Siguiente`.
- Es mostrarà el diàleg per a seleccionar l'element a exportar. Tria la classe `Student.cs` i prem novament `Siguiente`.
- Apareixerà un diàleg per a seleccionar quines referències es desitgen per defecte. **No marques cap** i polsa `Siguiente`.
- Finalment, un nou diàleg es mostrarà per indicar les opcions de la plantilla. Canvia el nom de la plantilla per `GestAcaClassDomain` i prem `Finalitzar`.

En aquest moment **és necessari tancar Visual Studio i tornar-lo a obrir per a què Visual carregue la plantilla creada**. La Figura 8 mostra de forma resumida els passos a seguir.

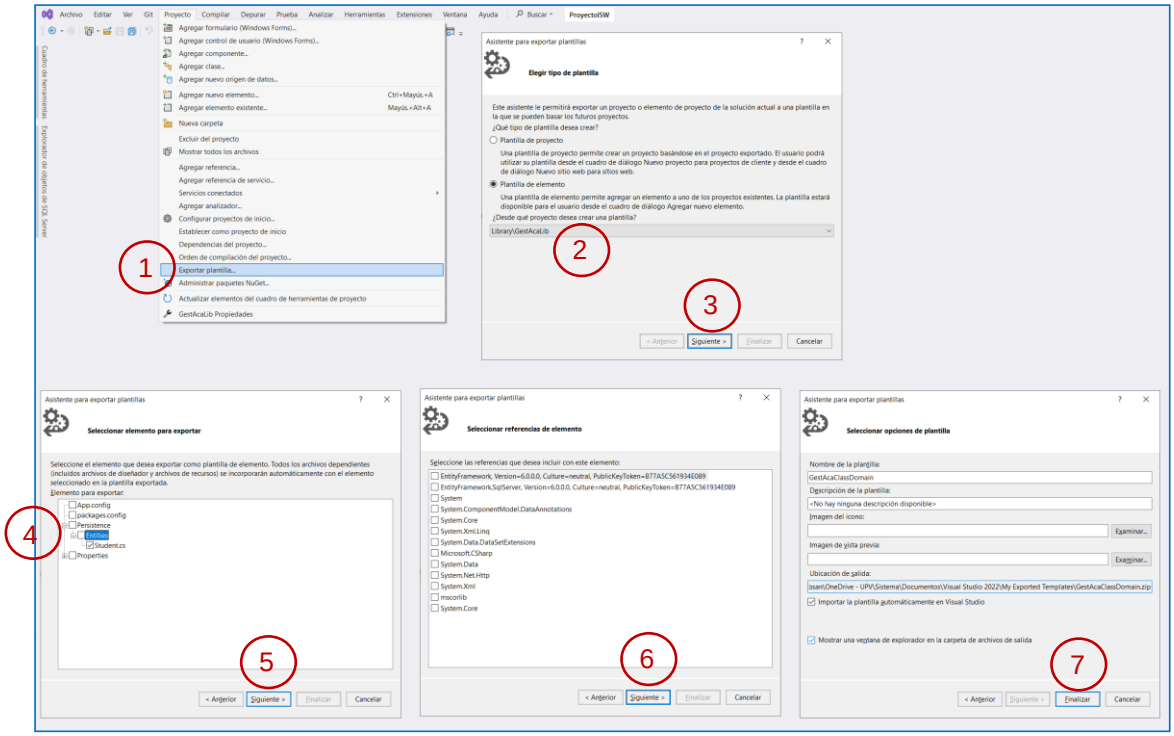

Figura 8. Creació d'una plantilla per a les classes del domini

4.3 Afegir la resta de classes del model en Persistence/Entities

Després d'obrir el projecte en Visual Studio, **el responsable de l'equip** crearà la resta de classes utilitzant la plantilla del punt anterior, repetint per a cadascuna d'elles els següents passos:

- Situar-se sobre la carpeta de solucions *Persistence/Entities.*
- Obrir el menú contextual amb el botó dret del ratolí.
- Seleccionar `Agregar > Nuevo Elemento`.
- Triar l'opció GestAcaClassDomain i a l'apartat nom, indicar el nom de la classe a crear (veure Figura 9).
- Polsar `Agregar`

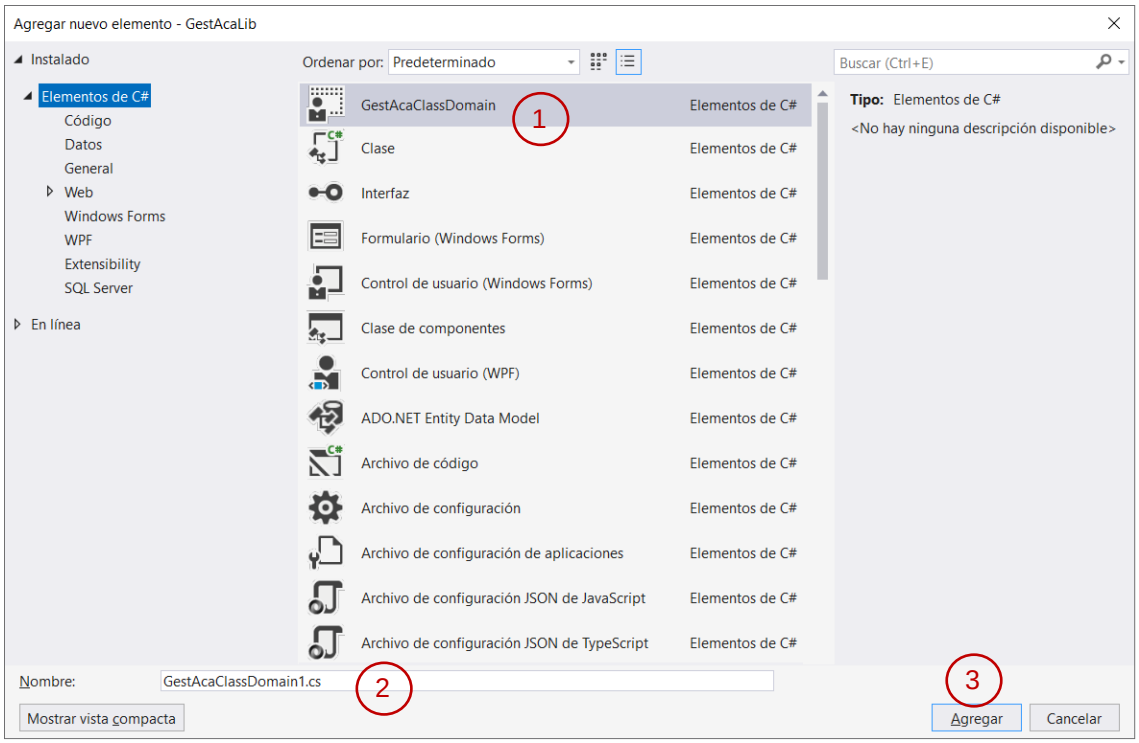

Figura 9. Utilitzar la plantilla per a crear las classes

Quan acabeu, haureu de tindre com a resultat el mateix contingut que es pot veure a la Figura 10.

Novament, és un bon moment per a guardar els canvis al projecte i sincronitzar-los amb la resta de membres de l'equip.

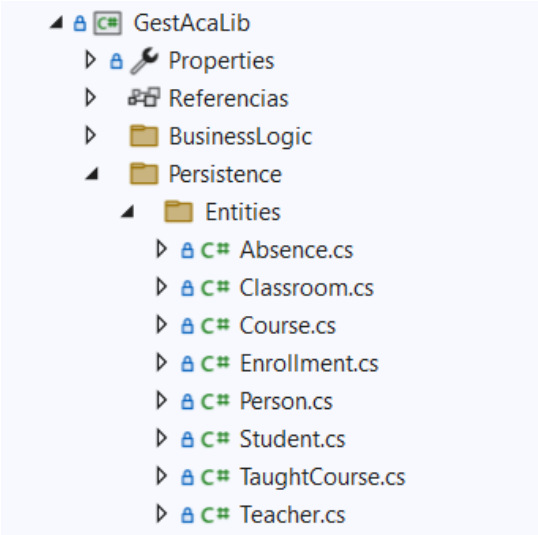

Figura 10. Classes buides creades en Persistence, Entities

4.4 Afegir les classes a BusinessLogic.Entities

Tal i com s'ha indicat anteriorment, anem a implementar les classes utilitzant **classes parcials,** de forma que el codi corresponent a la capa de persistència estarà situat a les classes que es troben dins de la carpeta *Persistence/Entities*, mentre que la resta del codi es situarà a *BusinessLogic/Entities.* **El responsable de l'equip** va a copiar totes les classes creades recentment (que encara estan sense codi) a la carpeta *Persistence/Entities.* Per tant, només ha de triar totes les classes i, des del menú contextual, triar l'opció copiar. Després, es situa a *BusinessLogic/Entities* i des del menú contextual, selecciona l'opció pegar.

Novament, ha de fer un *commit* del projecte i inserir-lo al repositori remot.

4.5 Afegir el tipus numerat

Si el diagrama de classes incloguera algun tipus numerat també deurien crear-lo. En el cas d'estudi GestAca no hi ha cap enumerat, per lo tant, no hem de fer res mes. Si hi haguera un enumerat, per exemple, un enumerat denominat Authorized, amb tres possibles valors (Yes, No, Pending), hauríem de crear el tipus enumerat `Authorized`. Per tal de fer-ho, el responsable es situaria dins de la carpeta *Persistence/Entities,* i, des del menú contextual, selecciona l'opció `Agregar > Nuevo Elemento`. En aquest cas, has de seleccionar l'opció `Archivo de código` i anomenar-lo, Authorized.cs (veure Figura 11).

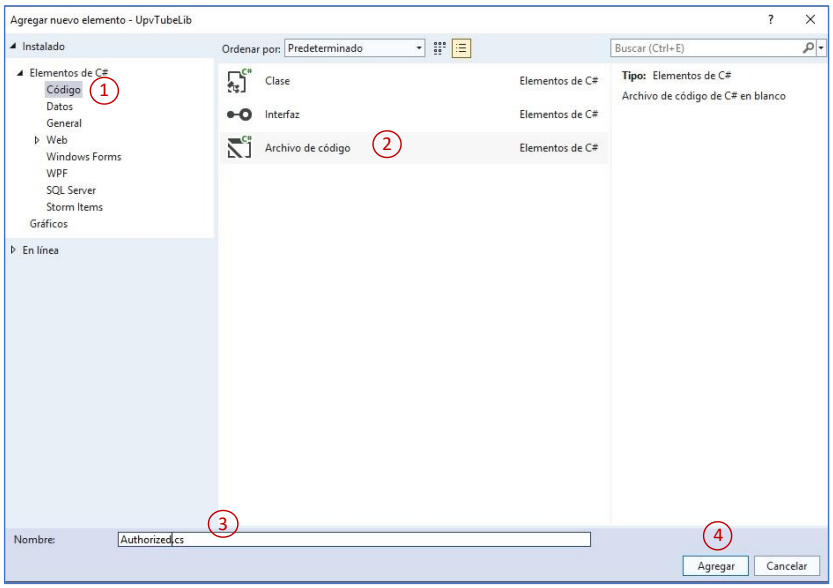

Figura 11. Agregar un arxiu de codi per a implementar el tipus enumerat

El codi de `Authorized` deuria d'editar-se afegint els valors del tipus enumerat així:

```
using System;
namespace GestAca.Entities
{
    public enum Authorized : int
    {
       Yes,
       No,
       Pending,
    }
}
```

Ara ja tenim tots els elements necessaris per començar a completar el codi de les classes. En aquest moment, la solució ha de tindre el mateix aspecte que el de la Figura 12.

El responsable guarda els canvis i els sincronitza amb el repositori remot. La resta de companys ha de sincronitzar els seus repositoris i extraure tots els canvis realitzats.

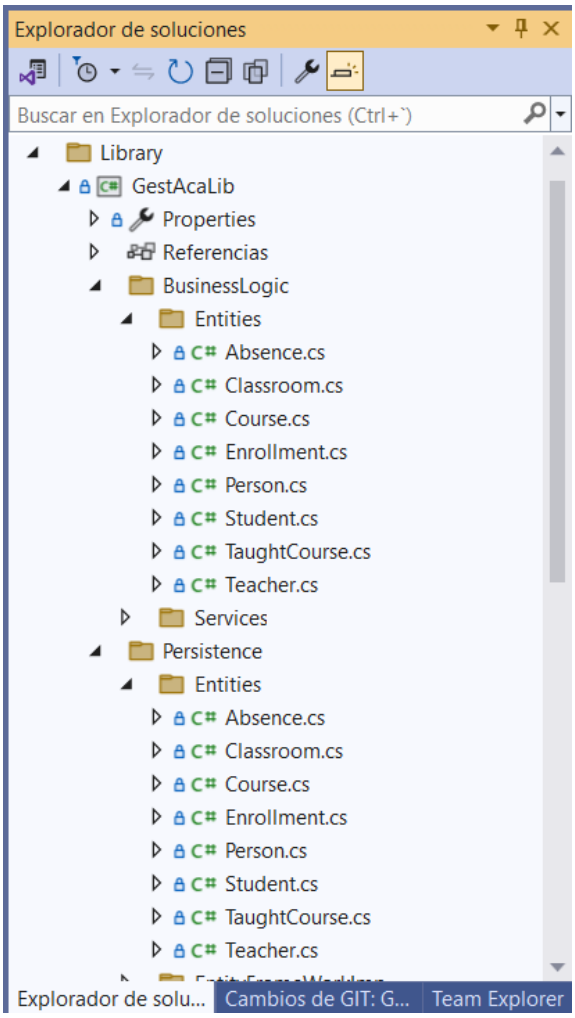

Figura 12. Estat de la solució després de crear totes les classes buides

5. Implementar l'aspecte de persistència de les classes

A partir d'aquest punt, **tots els membres de l'equip poden participar en la implementació**, repartint el treball per a què cada membre treballe de forma independent en cada fitxer per evitar conflictes.

Es recomana guardar els canvis i sincronitzar-los per cada classe finalitzada.

L'aspecte de persistència l'anem a implementar a les classes situades a *Persistence/Entities.* A cadascuna d'elles heu d'afegir les propietats indicades al disseny més les propietats necessàries per a representar les associacions del diagrama, seguint les pautes explicades al tema 5 de teoria. En aquestes pautes s'explica com hem de fixar-nos en les cardinalitats màximes de les associacions per saber si una associació es converteix en una propietat (cardinalitat màxima d'1) o en una col·lecció (cardinalitat màxima n). A més, hem de fixar-nos en les restriccions de navegabilitat del diagrama de disseny, en cas que no totes les associacions foren bidireccionals.

D'altra banda, les propietats que apareixen al disseny (els ***atributs***) es declararan com a **public**. Les propietats per a representar les ***relacions*** les declararem com a **public virtual**, ja que és un requisit que necessitarem per a implementar la capa de persistència.

EXEMPLE – Cas Estudi TarongISW

Anem a gastar un exemple (cas d'estudi TarongISW) per a il·lustrar la implementació de les propietats de les classes aplicant les pautes de disseny. En concret, a partir del diagrama de classes de disseny de la Figura 13, s'implementaran les classes *Contract* i *Temporary.* De forma similar, l'equip tindrà que implementar les classes del cas d'estudi Magazine (cada membre de l'equip es responsabilitzarà de implementar unes classes.)

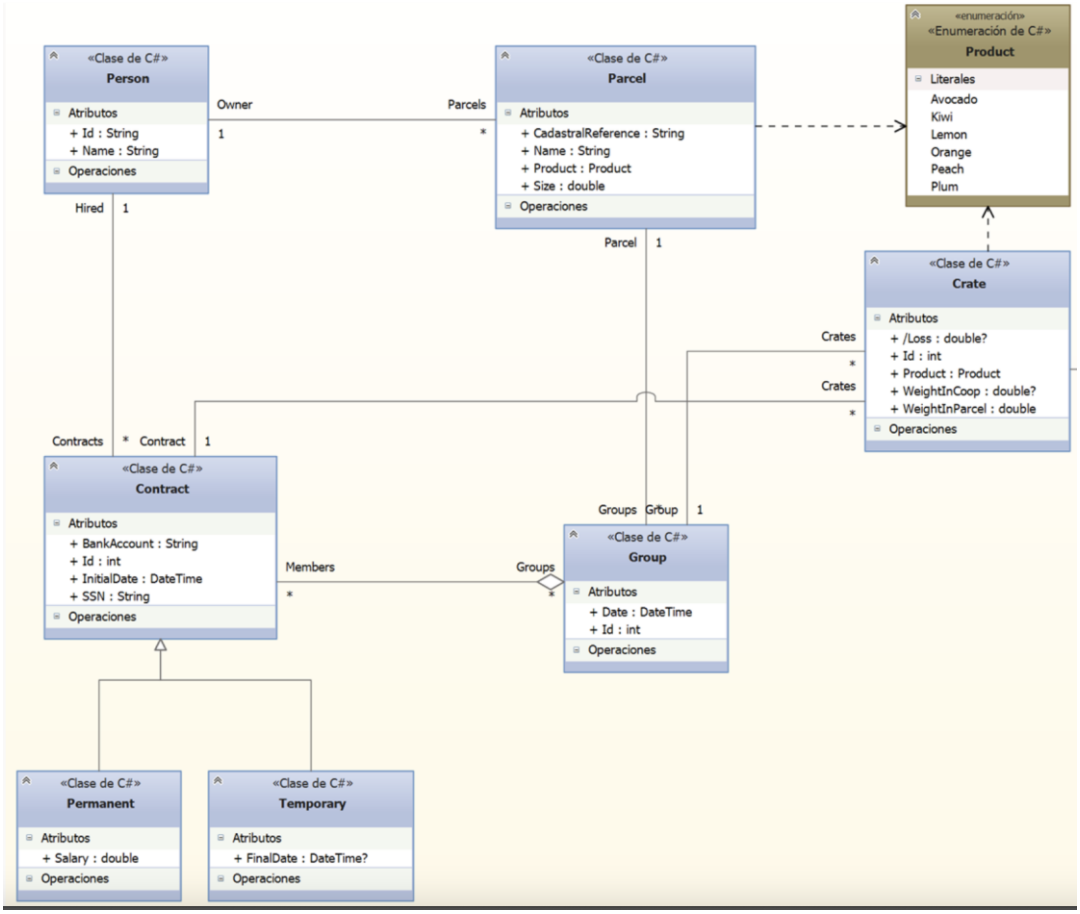

Figura 13. Diagrama de classes d'exemple- Cas d'estudi TarongISW

5.1 Implementació de la classe *Contract*

La classe *Contract* és una generalització que representa els elements comuns de dues tipus de contractes, representats per la classe *Permanent* i la classe *Temporary.* Així, en el codi apareix una propietat per cadascuna de les propietats descrites en el model. A més, la classe *Contract* es relaciona amb les classes *Group i Crate* mitjançant associacions amb cardinalitat màxima n que s'implementen amb les col·leccions *Groups i Crates,* respectivament. Fixeu-vos que el nom de les col·leccions coincideixen amb els rols de la corresponent relació en el model. També es relaciona amb la classe *Person* mitjançant una associació amb cardinalitat 1, que s'implementa amb la propietat *Hired* de tipus *Person*.

Observeu que primer s'han afegit les propietats pròpies de la classe i després les propietats derivades de les relacions amb altres classes segons el model.

```
namespace TarongISW.Entities
{
    public partial class Contract
    {
        public string BankAccount
        {
                get;
                set;
        }

        public int Id
        {
                get;
                set;
        }


         public DateTime InitialDate
        {
                get;
                set;
        }


        public string SSN
        {
                get;
                set;
        }


        public virtual Person Hired
        {
                get;
                set;
        }

        public virtual ICollection<Crate> Crates
        {
                get;
                set;
        }

        public virtual ICollection<Group> Groups
        {
                get;
                set;
        }

}
```

5.2 Implementació de la classe *Temporary*

La classe *Temporary* és una especialització de la classe *Contract*, per la qual cosa implementarem *Temporary* com a una classe que hereta de *Contract* (`public partial class Temporary : Contract`). Esta classe no es relaciona amb ninguna altra, per lo que únicament tenim que aferir una propietat per cadascuna de les propietats descrites al model, amb el tipus indicat.

En aquest cas, soles tenim una propietat a la que apareix un "?" després del tipus de dades:

```
FinalDate: DateTime?
```

Açò permet que el valor de l'atribut el tipus de dades del qual no és un objecte pot admetre valors nuls. Les referències a objectes poden ser nul·les, però no els valors (tipus primitius i struct). Aquesta notació permet fer-los nuls també[1]. En el nostre exemple, la data de finalització de contracte (`FinalDate`) és de tipus `DateTime`, que és un *struct*. Aquest atribut pot ser que no tinga un valor de data vàlida fins que finalitzarà el contracte temporal. Per a permetre un valor nul en aquesta data es defineix amb el tipus `DateTime?`. De manera similar, quan al constructor s'utilitza un paràmetre d'aquest tipus, s'ha de definir del tipus `DateTime?`, i així se li pot passar el valor `null`. Seguidament adjuntem el codi de la classe Temporary.

```
namespace TarongISW.Entities
{
    public partial class Temporary : Contract
    {
        public DateTime? FinalDate {
            get;
            set;
        }

    }
}
```

Per a comprovar si una variable definida d'aquesta manera té un valor `null`, s'utilitza `HasValue`, i per accedir al valor, `Value`, com il·lustra l'exemple següent:

```
            DateTime? EndDate = null;


            if (EndDate.HasValue)
                Console.WriteLine(EndDate.Value);
            else
                Console.WriteLine("La data de cancel.lació no té valor.");
}
```

(Aquest codi no ha d'incloure's ara, solament si es necessita durant la implementació de la lògica de l'aplicació)

6. Implementar els constructors de les classes en BusinessLogic

El codi dels constructors de les classes ha d'implementar-se als fitxers de les classes allotjades en *BusinessLogic/Entities*. Novament, tots els membres de l'equip poden col·laborar amb la implementació, però han de dividir-se el treball de manera que **cadascú treballe en una classe diferent per tal de no generar conflictes**.

Es recomana fer *commits* i sincronitzar per cada classe finalitzada.

Totes les classes del nostre sistema **tindran dos constructors**, un sense paràmetres i un altre amb tots els paràmetres necessaris. El constructor sense paràmetres simplement inicialitzarà les col·leccions que tinga la classe creant **col·leccions buides**. L'altre constructor (amb paràmetres), a més d'inicialitzar les col·leccions, ha de rebre el valor de totes les propietats que necessiten ser inicialitzades.

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

IMPORTANT:

No afegirem al constructor l'atribut Id si és de tipus Integer. Usarem **EntityFramework (EF)** per a la persistència de la informació, i EF **s'encarrega de donar valor als atributs ID de tipus numèric** de manera automàtica en el moment en el que persisteix l'objecte per primera vegada. Per tant, no hem de donar valor de forma manual a aquesta propietat, ja que és EF qui li assignarà el valor en quant l'emmagatzeme per primera vegada.

L'ordre dels paràmetres al constructor serà: en primer lloc, **els paràmetres propis en ordre alfabètic.** Després, **els objectes que ha de rebre per a afegir-los a les propietats afegides a causa de les associacions** de la classe amb altres, també en ordre alfabètic. En el cas de les classes que **hereten** d'altres, en primer lloc passarem el bloc dels paràmetres que rep la classe pare, ordenats alfabèticament, després el bloc amb els paràmetres propis (també ordenats) i, darrerament, les propietats afegides a causa de les associacions.

Cal recordar que per al cas de les classes que tenen atributs "nullables" (amb "?"), el constructor amb paràmetres ha de declarar també el **tipus de dades amb "?"** per a què hi haja concordança de tipus.

Com a exemple, seguidament es proporcionarà el codi de les mateixes classes utilitzades d'exemple al punt anterior: `Contract i Temporary`.

6.1 Implementació dels constructors de la classe *Contract*

`Contract` té dues col·leccions, `Crates` i `Groups`, que han d'inicialitzar-se en el constructor sense paràmetres. El constructor amb paràmetres rep tots els paràmetres necessaris per a instanciar les seues propietats, excepte `Id`, ja que és de tipus `int`. **Fixeu-vos en què són rebuts en ordre alfabètic.** La cardinalitat mínima per a les relacions amb Group i Crate és 0 pel que no és necessari assignar cap element inicialment a les col·leccions i, per tant, no és necessari passar al constructor cap paràmetre. És a dir, no és necessari assignar inicialment a un contracte un grup o un calaix. La cardinalitat amb `Person` és 1 pel que, quan es crea un contracte, aquest ha de relacionar-se necessàriament amb una persona . Per a això hem de passar el corresponent valor al constructor en el paràmetre `hired`. El codi resultant seria aquest:

```
namespace TarongISW.Entities
{
    public partial class Contract
    {
        public Contract()
        {
                // Col.leccions
                Crates = new List<Crate>();
                Groups = new List<Group>();
        }

        public Contract(string bankAccount, DateTime initialDate, string SSN, Person
hired)
        {

                // No es dona valor a Id perque ho farà EF
                this.BankAccount = bankAccount;
                this.InitialDate = initialDate;
                this.SSN = SSN;

                // Relacions a 1
                Hired = hired;

                // Col.leccions
```

```
            Crates = new List<Crate>();
            Groups = new List<Group>();
        }
}
```

6.2 Implementació dels constructors de la classe Temporary

`Temporary` hereta de `Contract`, per la qual cosa els seus constructors cridaran als constructors de la seua classe pare per a què realitze la inicialització de les propietats heretades, utilitzant la sentència `base()`. A més, al constructor amb paràmetres, passarem en primer lloc i en ordre alfabètic els paràmetres que rep per instanciar les seues propietats heretades i, en segon lloc, els paràmetres propis (també en ordre alfabètic).

Si hi ha atributs que es seu valor no es coneix en el moment de crear l'objecte no es passaran com a paràmetre al constructor. Són els que estan marcats en el diagrama de disseny amb el símbol ?. En aqueix cas, dins del constructor podran inicialitzar-se a un valor per defecte o nul. En la classe Temporary, el valor de FinalDate no es coneix en el moment de instanciar l'objecte i s'inicialitza a null en el constructor.

```
namespace TarongISW.Entities
{
    public partial class Temporary
    {
        public Temporary() {

        }

       public Temporary(string bankAccount, DateTime initialDate, string SSN,
Person hired):base(bankAccount, initialDate, SSN, hired)
        {

        }
    }
}
```

6.3 Reutilitzar codi amb this

En les classes amb col·leccions, és necessari inicialitzar-les tant en el constructor sense paràmetres com en el constructor amb paràmetres. Per això, és possible reutilitzar codi cridant al constructor sense paràmetres des del constructor amb paràmetres, tal com es mostra en el següent exemple. Destacar que això no és possible quan existeix herència, ja que s'ha de cridar a `base()`. El codi de la classe `Contract` utilitzant `this()` seria el següent, on pots veure que ja no apareix en el constructor amb paràmetres la inicialització de les llistes: `namespace TarongISW.Entities`

```
namespace TarongISW.Entities
{
    public partial class Contract
    {
        public Contract()
        {
                // Colecciones
                Crates = new List<Crate>();
                Groups = new List<Group>();
        }

        public Contract(string bankAccount, DateTime initialDate, string SSN, Person
hired): this()
        {
                // No se le da valor a Id porque se lo dará EF
                this.BankAccount = bankAccount;
                this.InitialDate = initialDate;
                this.SSN = SSN;

                // Relaciones a 1
                Hired = hired;

        }
}
```

7. Treball a entregar

En finalitzar les dues sessions, els grups hauran d'haver acabat la implementació de totes les classes del model, així com el tipus enumerat. Les propietats de les classes han d'estar implementades als fitxers situats a *Persistence/Entities*. Els constructors de les classes han d'estar implementats a *BusinessLogic/Entities.* A més, per cada classe hem de tindre dos constructors:

- Un amb paràmetres, que en cas de què la classe tinga col·leccions les inicialitze.
- Un altre amb els paràmetres necessaris, on aquestos paràmetres respecten l'ordre establert a l'apartat 6. A més, aquestos constructors també han d'inicialitzar les col·leccions que continga.

Finalment, el codi de cada equip haurà de passar una sèrie de **test**, **per la qual cosa, si no es segueixen les pautes anteriors, el codi no passarà les proves i no podrà continuar amb la implementació de la capa de persistència.**