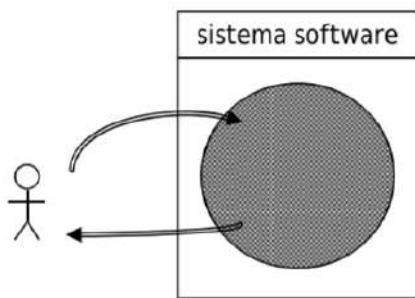
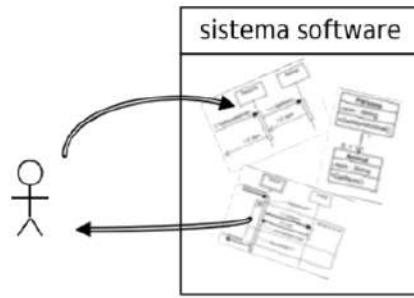


Visió del Disseny Orientat a Objectes

Especificació



Disseny



Classes no associatives

- Les classes d'especificació són compatibles, per tant, es mantenen a disseny amb la mateixa forma.

Especificació

```
class Persona
{
    -nom : String
    -edat : Enter
};
```

Disseny

```
class Persona
{
    -nom : String
    -edat : Enter
};
```

Codi (C++)

```
class Persona
{
    string nom;
    int edat;
};
```

- Especificació:
 - el sistema software es veu com una sola classe d'objectes que engloba tota la informació i totes les operacions.
- Disseny:
 - cada classe té les seves operacions de manipulació d'informació. Els objectes interactuen per satisfer les operacions del sistema.

Transformació del diagrama de classes

- El diagrama de classes d'especificació és més expressiu que el diagrama de classes de disseny, per tant, per caldrà:
 - Mantenir aquells elements que siguin compatibles:
 - Classes no Associatives
 - Generalitzacions/Especialitzacions Disjoint.
 - Associacions binàries
 - Atributs
 - Eliminar aquells elements que no siguin compatibles i reemplaçar-los per d'altres que sí que ho siguin:
 - Classes Associatives
 - Associacions N-àries
 - Altres casos de Generalització / Especialització
 - Classes Especials (com Data, Hora, ...)
 - Atributs derivats
 - Afegir aquells conceptes de disseny que no són necessaris a especificació.
 - Visibilitat
 - Àmbit
 - Excepcions

Generalitzacions/Especialitzacions disjoint

- Les herències disjoint són compatibles, per tant es mantenen igual

Especificació

```
class Persona
{
    -nom : String
    -edat : Enter
};

class Estudiant
{
    -notaMitja : Float
};

class Professor
{
    -salari : Enter
};

class Persona <|-- Estudiant
class Persona <|-- Professor
```

Disseny

```
class Persona
{
    -nom : String
    -edat : Enter
};

class Estudiant
{
    -notaMitja : Float
};

class Professor
{
    -salari : Enter
};

class Persona <|-- Estudiant
class Persona <|-- Professor
```

Codi

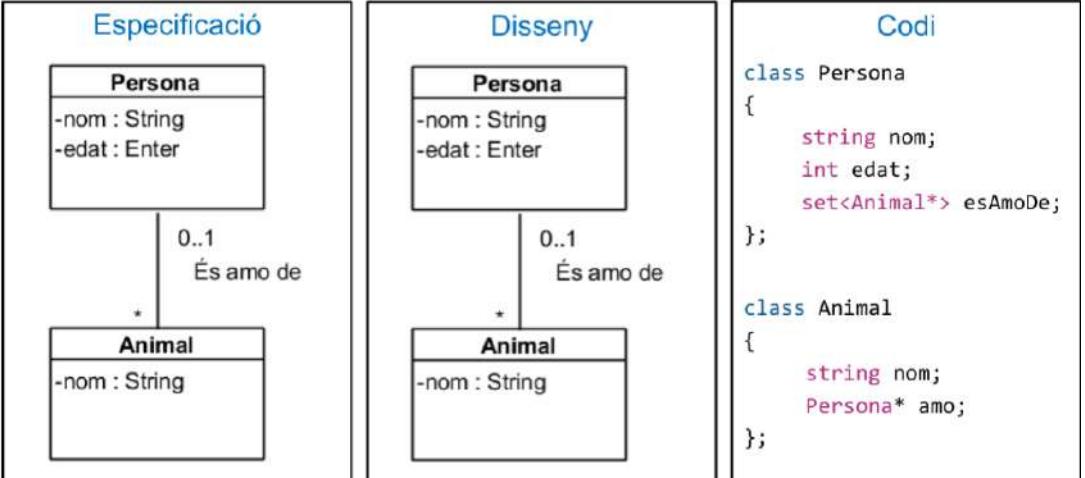
```
class Persona
{
    string nom;
    int edat;
};

class Professor: public Persona
{
    int salari;
};

class Estudiant: public Persona
{
    float notaMitja;
};
```

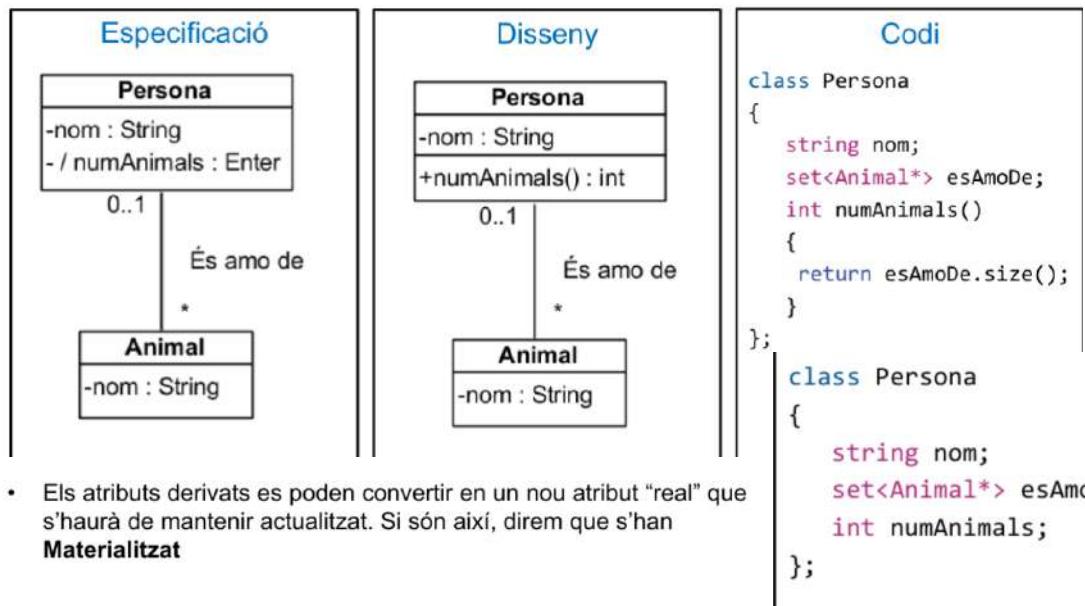
Associacions binàries

- Les associacions binàries es mantenen igual, tot i que a disseny els haurem d'afegir **navegabilitat** (al proper tema).



Atributs Derivats (I)

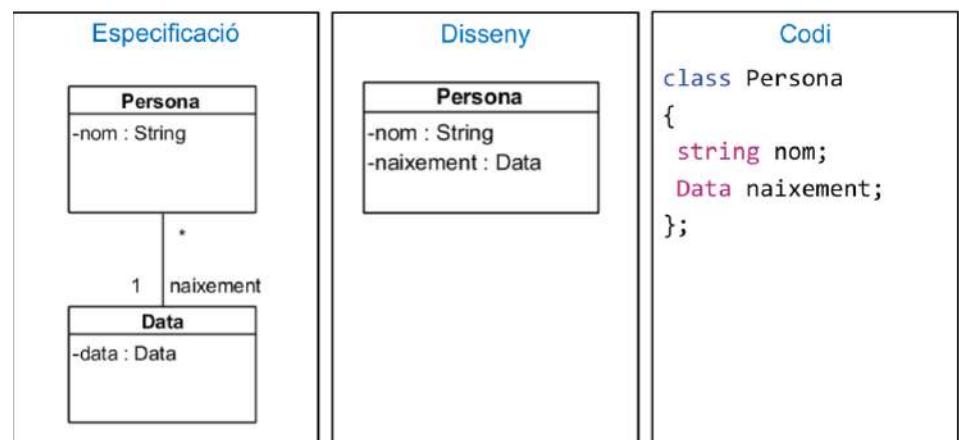
- Els atributs derivats es poden convertir en una funció nova que calcula el valor. Si són així, direm que són atributs **Calculats**



- Els atributs derivats es poden convertir en un nou atribut "real" que s'haurà de mantenir actualitzat. Si són així, direm que s'han **Materialitzat**

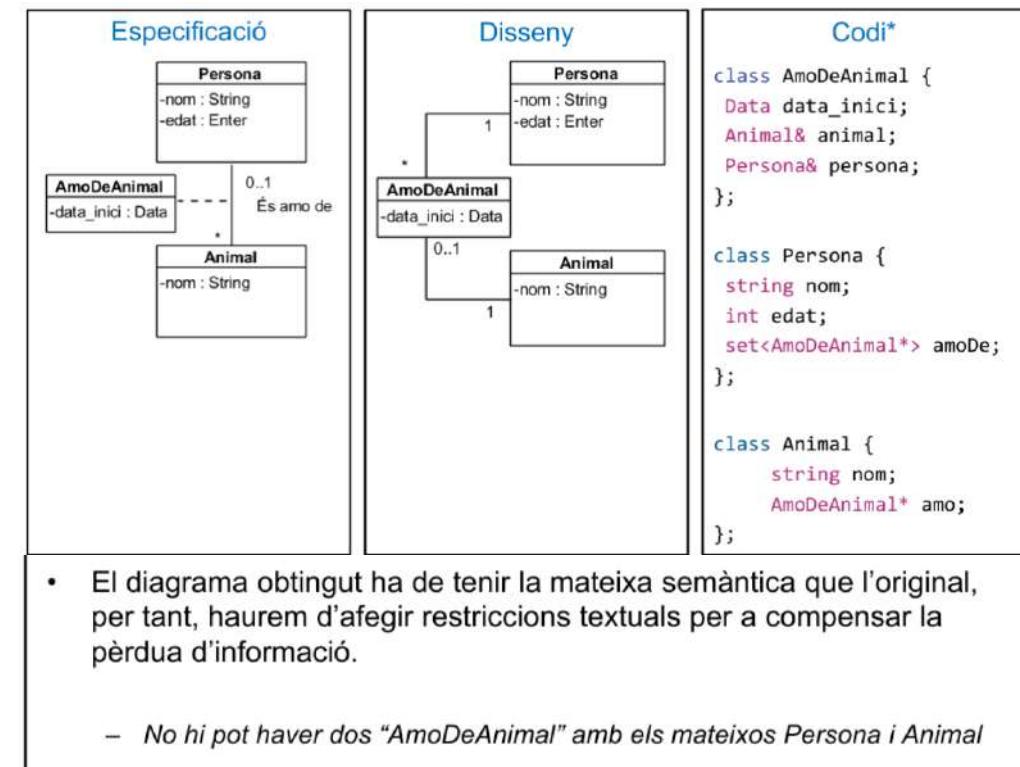
Classes Especials (Data, Hora, ...)

- Les classes especials esdevenen atributs amb el tipus corresponent, per tant, s'ha d'eliminar la classe original i afegir com a un atribut.



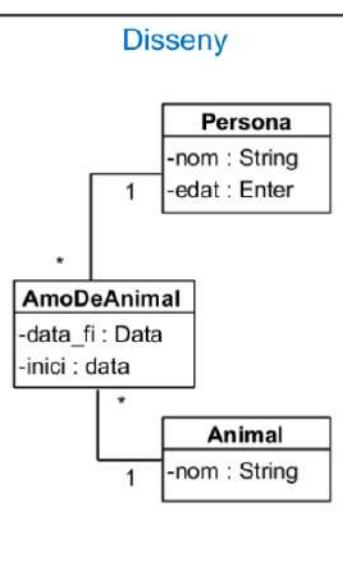
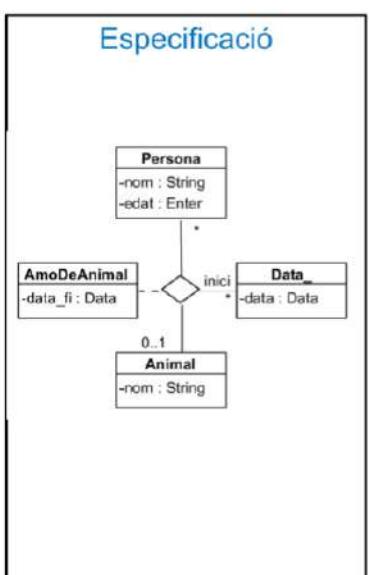
Classes Associatives (I)

- Les classes associatives es converteixen en classes no associatives i es relacionen amb les dues originals...



Associacions N-àries (I)

- Les associacions N-Àries s'han de canviar per una classe associativa relacionada amb les N originals...

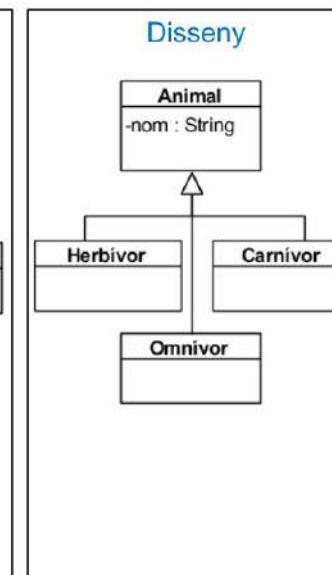
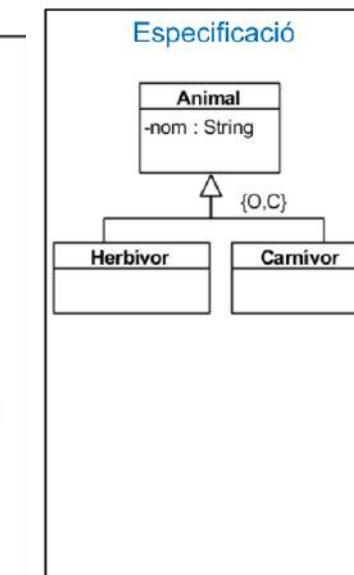


Codi*

```

class AmoDeAnimal {
    Data data_inici;
    Data data_fin;
    Animal& animal;
    Persona& persona;
}
class Persona {
    string nom;
    int edad;
    set<AmoDeAnimal*> amoDe;
}
class Animal {
    string nom;
    set<AmoDeAnimal*> amo;
}

```



Codi*

```

class Animal
{
    string nom;
}
class Carnivor:
    public Animal
{
}
class Herbivor:
    public Animal
{
}
class Omnivor:
    public Animal
{
}

```

- El diagrama obtingut ha de tenir la mateixa semàntica que l'original, per tant, haurem d'afegir restriccions textuales per a compensar la pèrdua d'informació (igual que amb les classes associatives).
 - *RT1: No hi pot haver dos “AmoDeAnimal” amb els mateixos Persona, Animal i Inici*
- També s'han de considerar les multiplicitats de cada un dels membres de la N-ària per a afegir noves restriccions
 - *RT2: Donada una persona i una Data, màxim pot esdevenir amo d'un Animal*
- I això pot provocar que algunes restriccions textuais siguin redundants.
 - *RT1 és redundant amb RT2, per tant, no s'ha d'afegir RT1*

Generalització / Especialització No Disjoint

- No hi ha una forma única de fer-ho depèn de cada situació. El producte cartesià és només una de les opcions, però n'hi ha més.

Visibilitat

- Defineix quins objectes tenen dret a consultar i modificar informació declarada en un diagrama de classes
- Pot ser de tres tipus
 - Pública (+)
 - Privada (-)
 - Protegida (#)
- Aplica a:
 - Atributs
 - Operacions
 - Rols
- A IES assumirem que, per defecte, els atributs i rols són privats, i les operacions són públiques.

Visibilitat - Públic

- Donat un element X d'una classe C
 - Si és públic, qualsevol que vegi C, veurà X

```
class Animal           void main()
{
    public:
        string nom;
    };
}                      {
    Animal animal;
    animal.nom = "Nix";
}
```

Visibilitat - Privat

- Donat un element X d'una classe C
 - Si és privat, només C veurà X

```
class Animal           void main()
{
    private:
        string nom;
    };
}                      {
    Animal animal;
    animal.nom = "Nix";
}
```

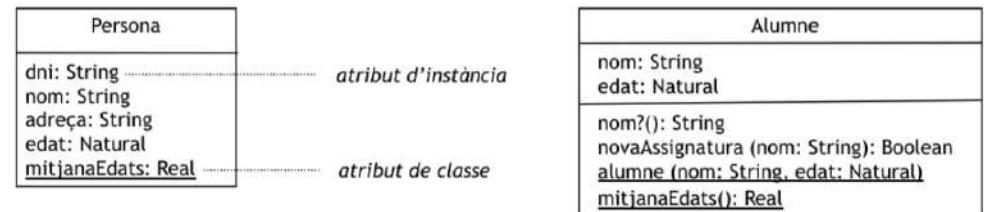
Visibilitat - Protegit

- Donat un element X d'una classe C
 - Si és protegit, només C o els seus descendents veuran X

```
class Animal {
    protected:
        string nom;
};
class Gos: public Animal {
public:
    void CanviaNom(string nouNom) {
        nom = nouNom;
    }
}                      void main()
{
    Gos animal;
    animal.nom = "Nix";
    animal.CambiaNom("Nix");
}
```

Àmbit

- Determina si els atributs o operacions són aplicables a objectes individuals o a la classe que defineix els elements
- Poden ser:
 - De classe (estàtic)
 - X està associat al C
 - D'instància (no estàtic)
 - X està associat als objectes de C
- Els atributs o operacions de classe (estàtics) es marquen subratllant el nom de la operació / atribut



Excepcions

- Els llenguatges de programació ens proporcionen diverses maneres de poder controlar els casos en els que la operació no s'ha pogut realitzar per causa d'algun problema.
- La més utilitzada és mitjançant l'ús **d'Excepcions**.

Com programaríem això?

```
Context: Sistema::FesLaResta(  
    num1: Enter Positiu,  
    num2: Enter Positiu  
): Enter Positiu
```

Pre: num1 >= num2

Body: result = num1 - num2;

Així!

```
Class Num1Menor: públic std::exception  
{}
```

```
unsigned int FesLaResta(  
    unsigned int num1,  
    unsigned int num2)  
{  
    if (num1 < num2)  
    {  
        throw Num1Menor();  
    }  
    return num1 - num2;  
}
```

Contractes de les Operacions a Disseny

- S'elimina la secció de precondicions
- S'afegeix una secció d'excepcions
 - Per cada possible problema que la operació pugui provocar crearem una nova excepció. La definiríem amb un **nom explicatiu i una descripció**.

Excepcions possibles

- Les excepcions que es poden produir en una operació poden ser dels següents tipus:
 - Violació d'una precondició
 - Violació d'una restricció textual
 - Violació d'una restricció gràfica
 - Violació d'una restricció del model (implícita)
- Per cada operació haurem de fer una anàlisi i veure quines es poden produir.

Violació d'una precondició

Especificació

Context: AltaPersona(
 nom: String,
 edat: Enter,
 nomCiutat: String)

Pre: nomCiutat existeix

Post: Es dóna d'alta una nova persona amb el nom, i l'edat introduïts, associada a la Ciutat nomCiutat

Disseny

Context: AltaPersona(
 nom: String,
 edat: Enter,
 nomCiutat: String)

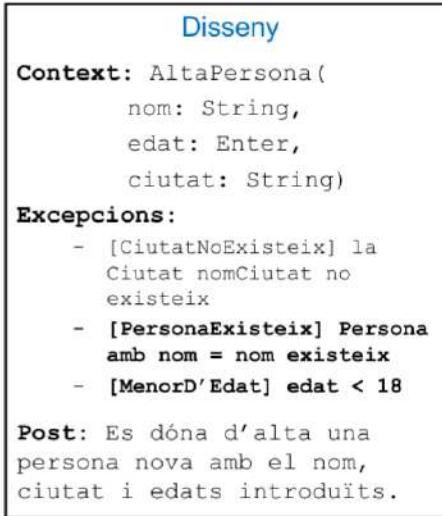
Excepcions:

- [CiutatNoExisteix] la Ciutat nomCiutat no existeix

Post: Es dóna d'alta una nova persona amb el nom, i l'edat introduïts, associada a la Ciutat nomCiutat

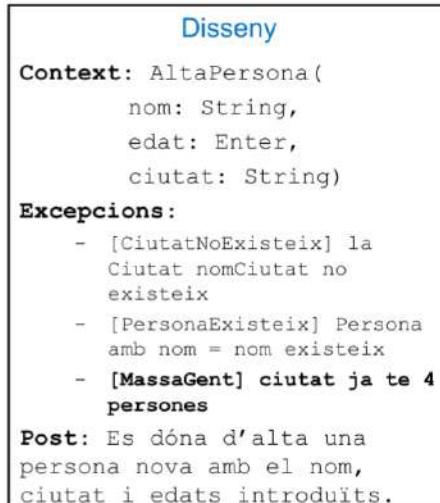
- Per cada Precondició que hi hagi a la nostra operació, haurem d'afegir una excepció nova

Violació d'una restricció textual



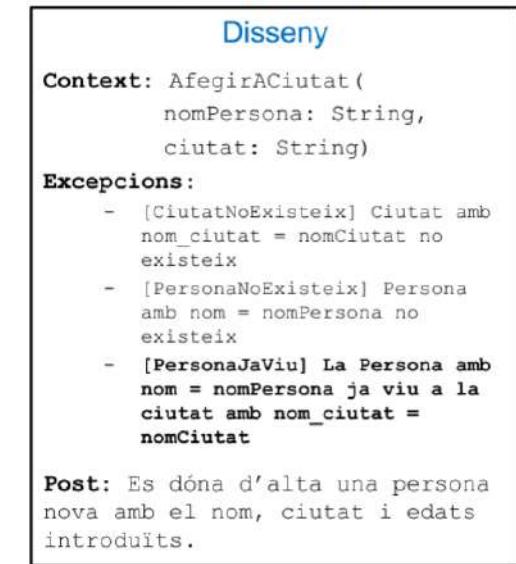
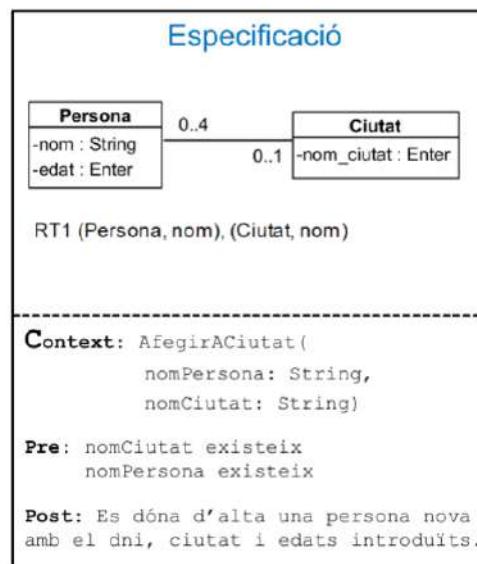
- Per cada restricció d'integritat que es pugui violar, haurem d'afegir una nova excepció.

Violació d'una restricció gràfica



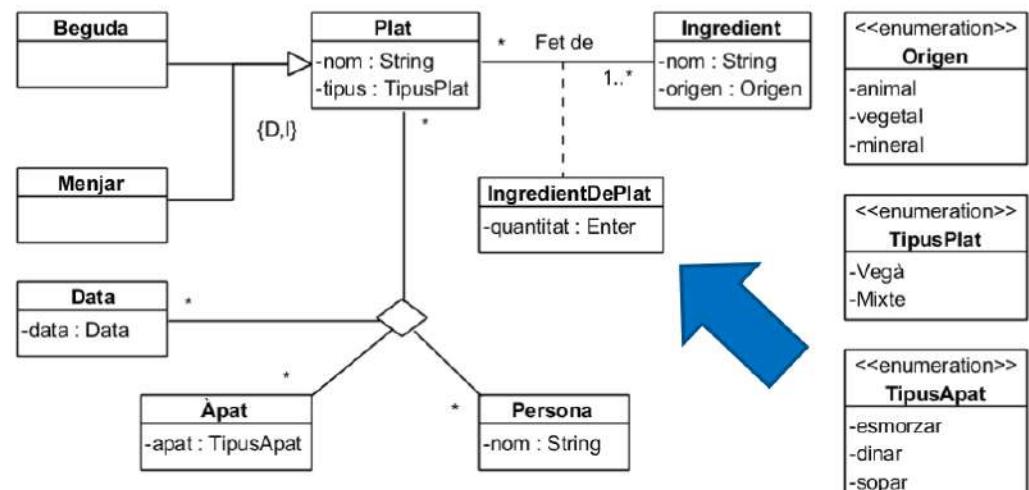
- Per cada multiplicitat que es pugui violar, cal una nova excepció

Violació d'una restricció del model (implícita)



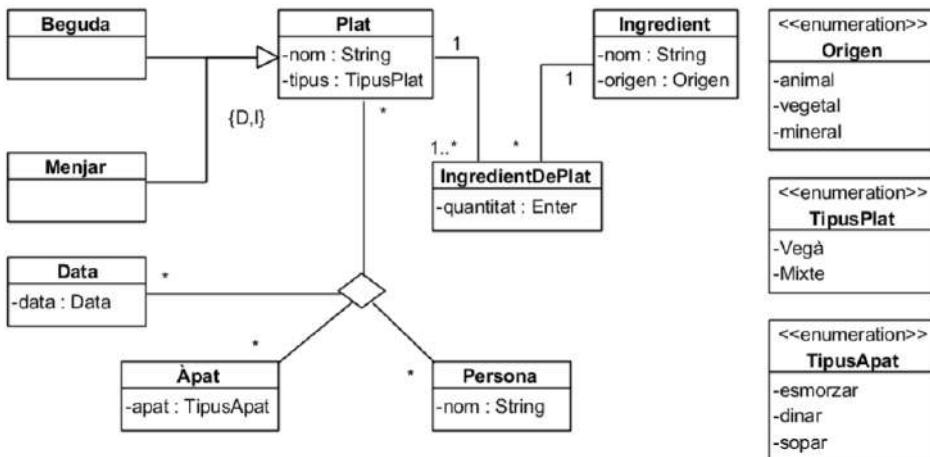
- Per cada multiplicitat que es pugui violar, cal una nova excepció

Exemple de traducció



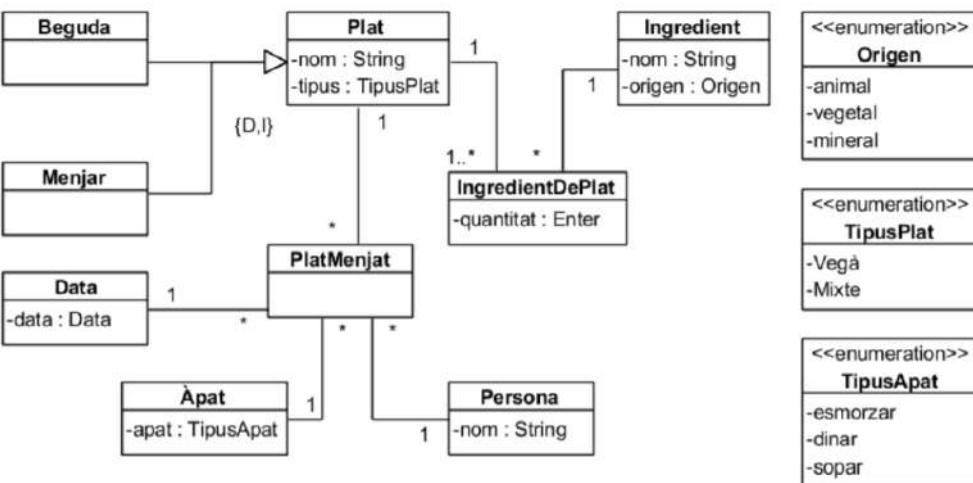
- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal

Exemple de traducció



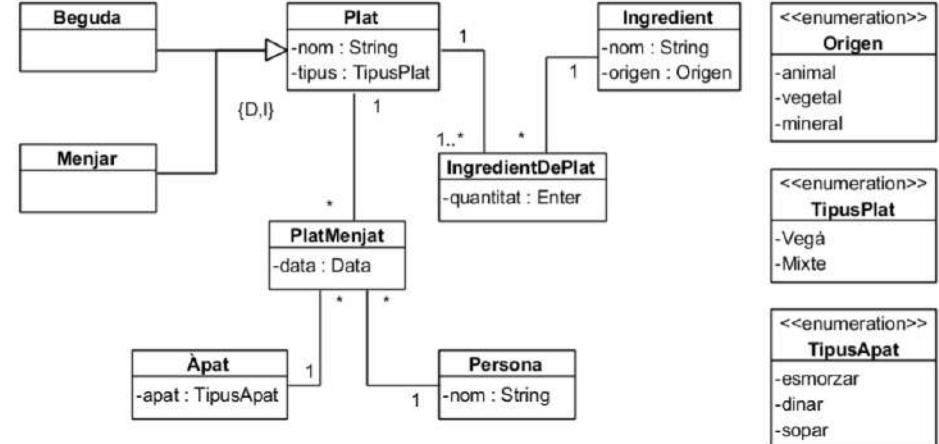
- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- RT3: No hi pot haver 2 IngredientDePlat amb els mateixos ingredient i plat

Exemple de traducció



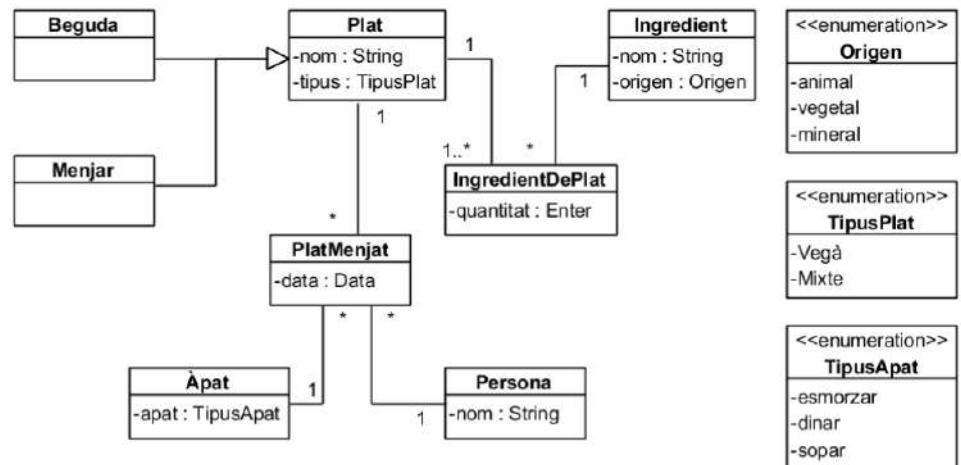
- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- RT3: No hi pot haver 2 IngredientDePlat amb els mateixos ingredient i plat
- RT4: No hi pot haver 2 PlatMenjat amb els mateixos Plat, Àpat, Data i Persona

Exemple de traducció



- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- RT3: No hi pot haver 2 IngredientDePlat amb els mateixos ingredient i plat
- RT4: No hi pot haver 2 PlatMenjat amb els mateixos Plat, Àpat, Persona i Data

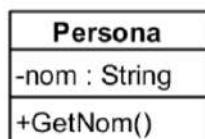
Exemple de traducció



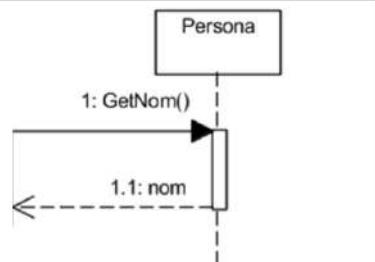
- RT1: Claus Externes (Plat, nom), (Ingredient, nom), (Àpat, tipusApat), (Persona, nom)
- RT2: Un plat vegà no pot contenir ingredients d'origen Animal
- RT3: No hi pot haver 2 IngredientDePlat amb els mateixos ingredient i plat
- RT4: No hi pot haver 2 PlatMenjat amb els mateixos Plat, Àpat, Persona i Data

Crides i Resultats

D. Classes

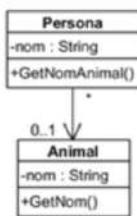


D. Seqüència

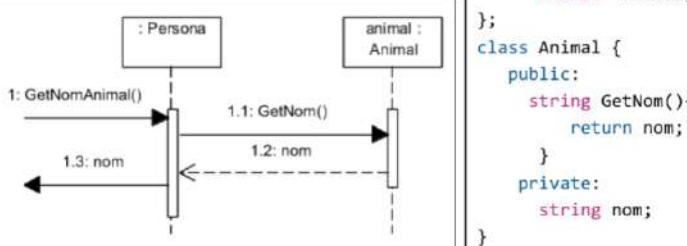


Crides entre Instàncies

D. Classes



D. Seqüència



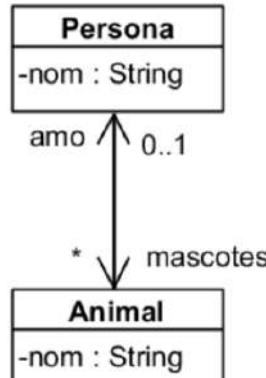
Codi

```

class Persona {
    public:
        string GetNom(){
            return nom;
        }
    private:
        string nom;
}
  
```

Navegabilitat bidireccional

D. Classes (Disseny)



Codi

```

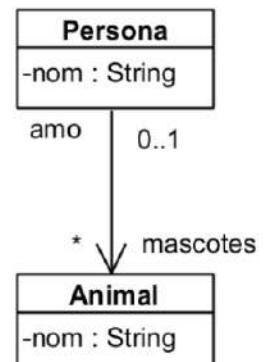
class Persona {
    string nom;
    set<Animal*> animal;
};

class Animal {
    string nom;
    Persona* amo;
}
  
```

- Les relacions navegables de forma bidireccional necessiten mantenir punters/referències a ambdues classes

Navegabilitat unidireccional

D. Classes (Disseny)



Codi

```

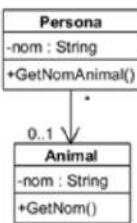
class Persona {
    string nom;
    set<Animal*> animal;
};

class Animal {
    string nom;
    Persona* amo;
}
  
```

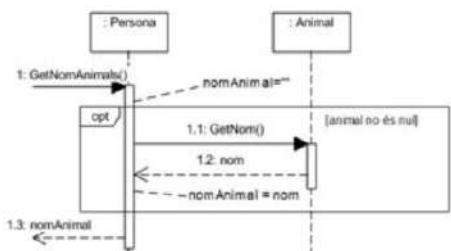
- Les relacions navegables de forma unidireccional necessiten mantenir punters/referències a una sola de les classes

Opcions / Condicionals

D. Classes



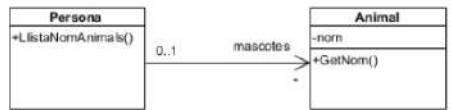
D. Seqüència



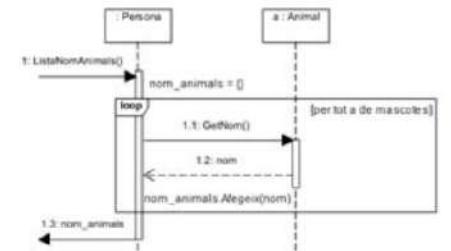
- La forma de representar *Opcions* o *Condicions* al diagrama de seqüència serà mitjançant l'ús dels frames de **opt** i **alt**, de la mateixa manera que es feia als diagrames de seqüència d'especificació.

Agregats

D. Classes



D. Seqüència



- Un agregat és una col·lecció d'objectes
- La forma de representar *iteracions sobre agregats* es fa mitjançant el frame de **loop** de la mateixa manera que es feia als diagrames de seqüència d'especificació

Agregats (recopilació)

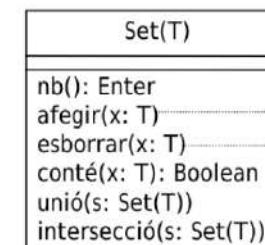
Un agregat és una col·lecció d'objectes

Sorgeixen en diversos contexts:

- Rols navegables amb multiplicitat més gran que 1
- Operacions que reben o retornen una col·lecció de valors
- Atributs multivaluats

En tots aquests casos, considerem l'agregat com un conjunt (Set)

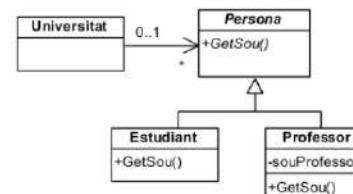
- S'hi poden aplicar operacions següents (i només aquestes):



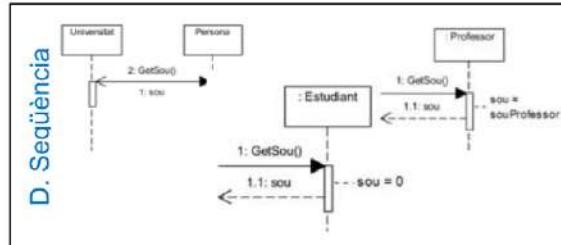
no fa res si *x* existeix
no fa res si *x* no existeix

Polimorfisme

D. Classes



D. Seqüència



- La forma de marcar el polimorfisme en el Diagrama de Seqüència és creant funcions amb el mateix nom dins de la mateixa estructura de Generalització / Especialització

- És un mecanisme dels llenguatges de programació que permet que, en el cas de que dues funcions es diguin igual en una jerarquia de classes, aleshores el sistema cridarà en temps d'execució a la funció que es trobi implementada al nivell més inferior a la jerarquia.

Codi

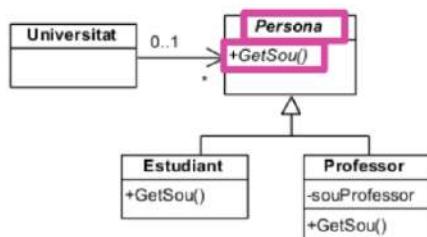
```

class Persona {
    public:
        virtual int GetSou() = 0;
    private:
        string nom;
    };
    class Estudiant: public Persona {
        public:
            int GetSou() override {
                return 0;
            }
    };
    class Professor: public Persona {
        int souProfessor;
        public:
            int GetSou() override {
                return souProfessor;
            }
    };
  
```

Polimorfisme: Elements Abstractes

- Quan una operació dins d'una classe no té cap mena de codi, aquesta s'ha de definir com a **abstracta**.
- Això provoca que:
 - Tots els seus descendents dins de la jerarquia estan obligats a tenir aquesta funció.
 - La pròpia classe esdevé **abstracta** i no es poden declarar instàncies que siguin exclusivament del seu tipus.
 - Una superclasse només pot ser abstracta si la seva especialització és *complete*.

Polimorfisme: Elements Abstractes

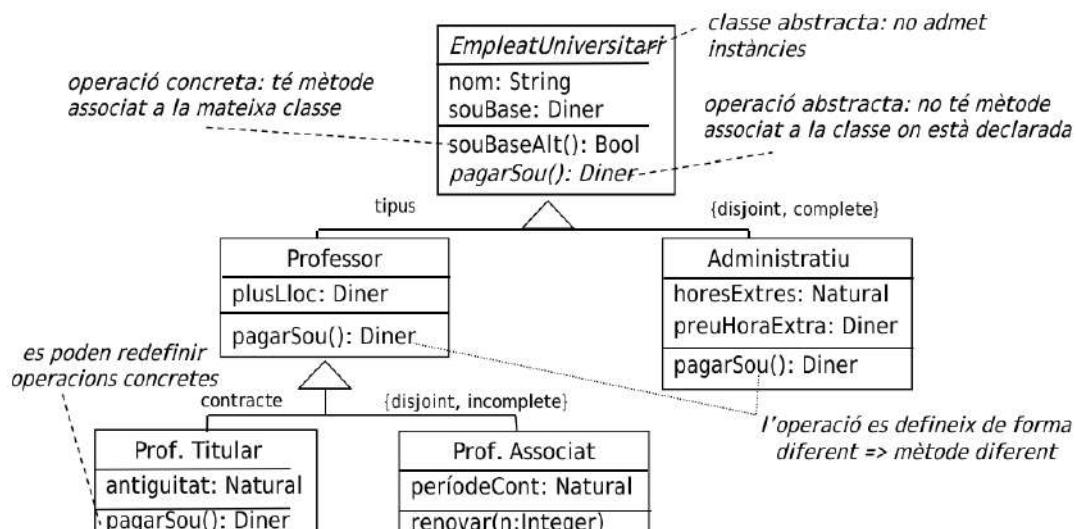


- La forma de marcar elements abstractes és posant el nom en cursiva
- A IES podeu posar una nota al costat de la classe/funció per a indicar-ho.
- Extra:** Aquesta és una manera de tenir especialitzacions / generalitzacions de tipus **Complete**.

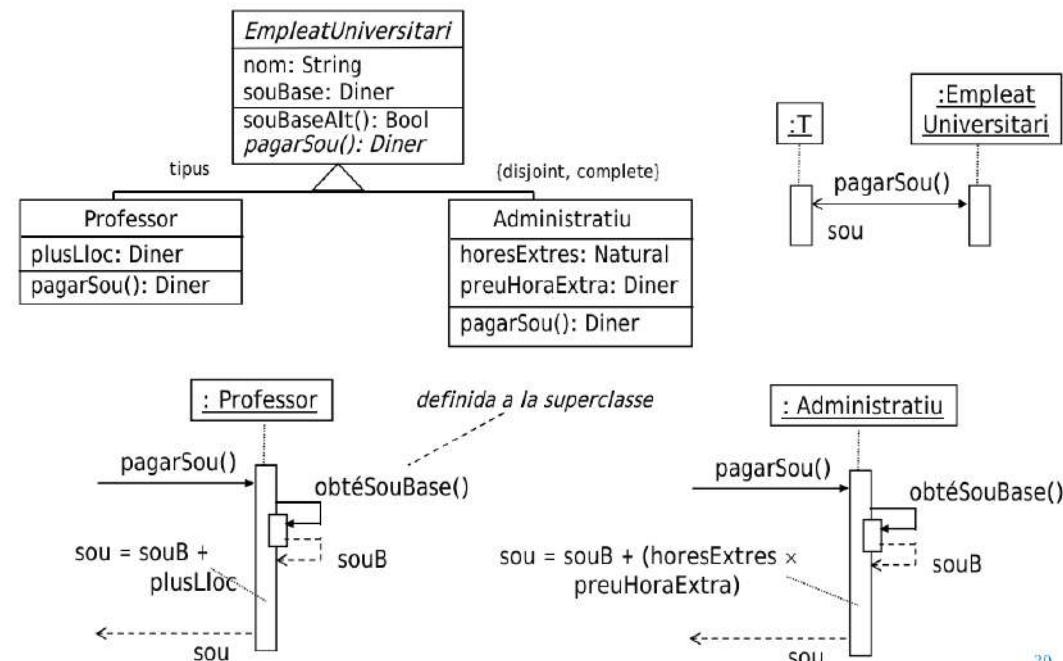
Polimorfisme

Operació polimòrfica:

- operació que s'aplica a diverses classes d'una jerarquia tal que la seva semàntica depèn de la subclasse concreta on s'aplica



Polimorfisme: vinculació dinàmica



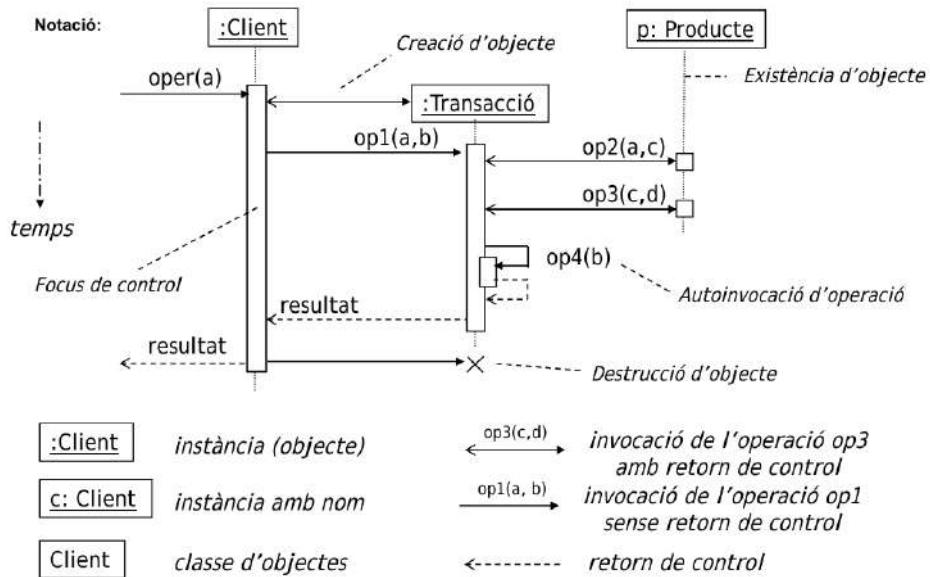
19

20

Creadores

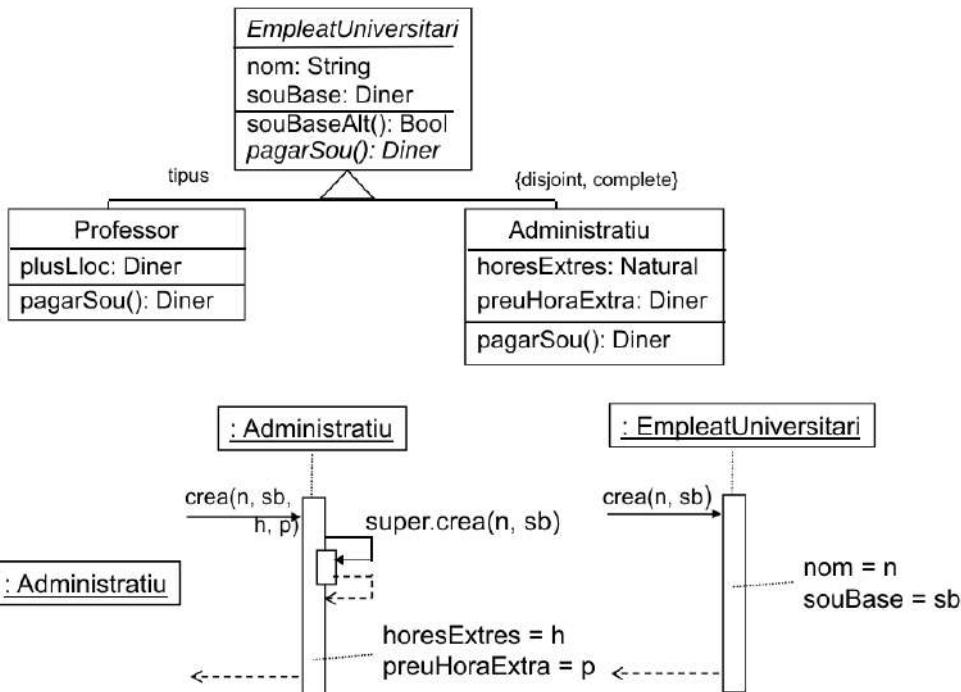
- La forma de crear instàncies als llenguatges de programació és, normalment, mitjançant l'ús d'unes funcions **estàtiques** especials anomenades **Creadores**
 - Una creadora rep com a paràmetres alguns (o tots, o cap) dels atributs de la classe que ha de crear i en retorna una instància.
 - En el cas de les creadores de subclasses d'una jerarquia, estan obligades a cridar a la creadora de la seva superclasse per a completar la seva construcció.

Diagrames de seqüència: sintaxi completa



23

Creació d'objectes en una jerarquia d'herència



Diagrames de seqüència: convencions (1)

Noms dels objectes: batejar-los quan calgui per identificar el seu origen

- si són resultat d'una operació, usar el mateix nom en el resultat i en l'objecte
 - si s'obtenen recurrent una associació amb multiplicitat 1, usar el nom del rol
 - si han arribat com a paràmetres, usar el nom dels paràmetres

Paràmetres de les operacions:

- cal indicar explícitament amb quins paràmetres s'invoquen les operacions

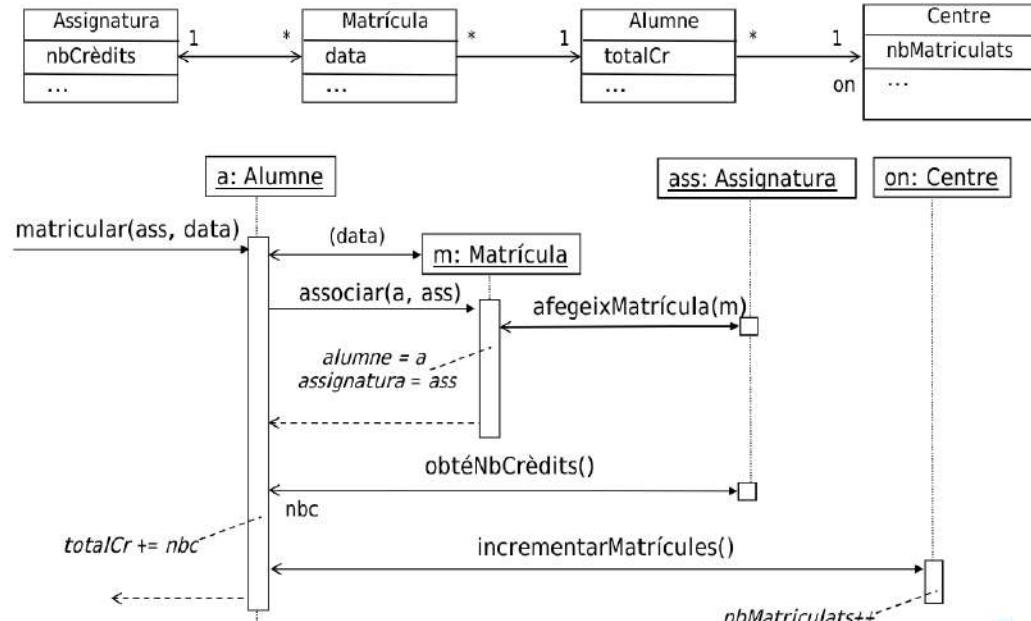
Resultats de les operacions:

- donar nom al resultat si surt en algun altre lloc del diagrama
 - també als valors retornats en els paràmetres

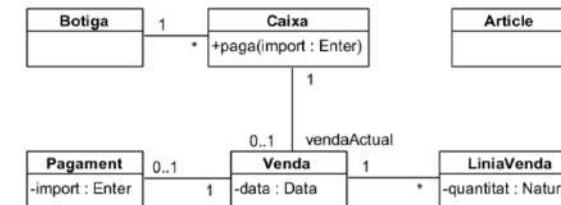
Comentaris:

- no deixar cap aspecte rellevant sense comentar
 - en particular, ha de quedar clar:
 - ✓ com es calculen els resultats de les operacions
 - ✓ Quins valors es passen com a paràmetres a les operacions creadores
 - ✓ com es modifiquen els valors dels atributs i pseudo-atributs de les classes
 - usar noms d'atributs, associacions, etc.
 - ✓ usar sintaxi Java per a operacions aritmètiques

Diagrams de seqüència: convencions (2)



Acoblament - Exemple



context Caixa:::paga(import: Enter)

pre: existeix vendaActual

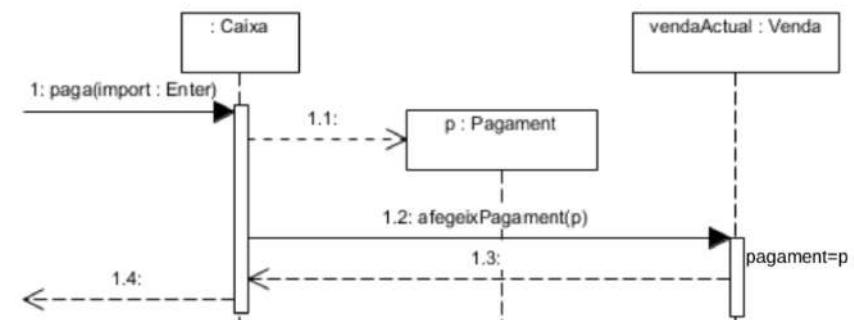
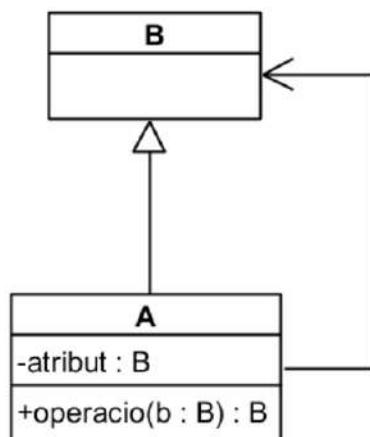
post: crea un pagament nou p amb p.import = import
associa la vendaActual amb el pagament

- Caixa crea un pagament i l'associa a Venda
- Introduceix un nou acoblament (dinàmic) entre Caixa i Pagament !

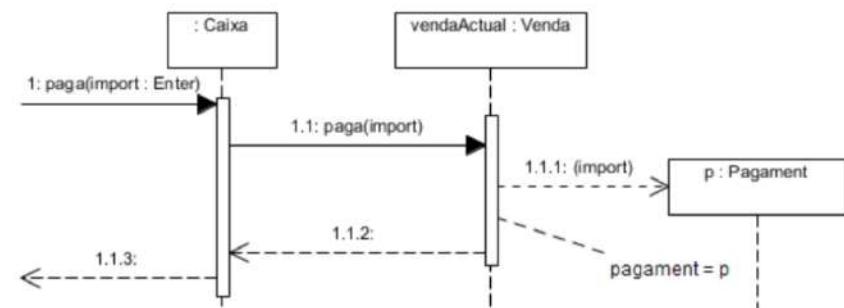
Acoblament

Direm que hi ha un acoblament entre la classe A i la classe B si:

- A té un atribut de tipus B
- A té una associació navegable amb B
- B és un paràmetre o el retorn d'una operació de A
- Una operació de A fa referència a un objecte de B
- A és una subclasse directa o indirecta de B
- ...



- Caixa delega a Venda l'operació de creació del pagament
- No introduceix nous acoblaments dinàmics



Acoblament – Llei de Demèter

Una operació només hauria d'invocar operacions (“parlar”) d'objectes accessibles des de *self* (“familiars”), que són:

- L'objecte que està executant l'operació (*self*)
- Un paràmetre rebut per l'operació
- Els valors dels atributs de l'objecte *self*
- Els objectes associats amb *self*
- Els objectes creats per la pròpia operació

Cohesió

Cohesió d'una classe és una mesura del grau de relació i de concentració de les diverses responsabilitats (atributs, associacions i operacions)

Informalment direm que una classe està ben cohesionada si:

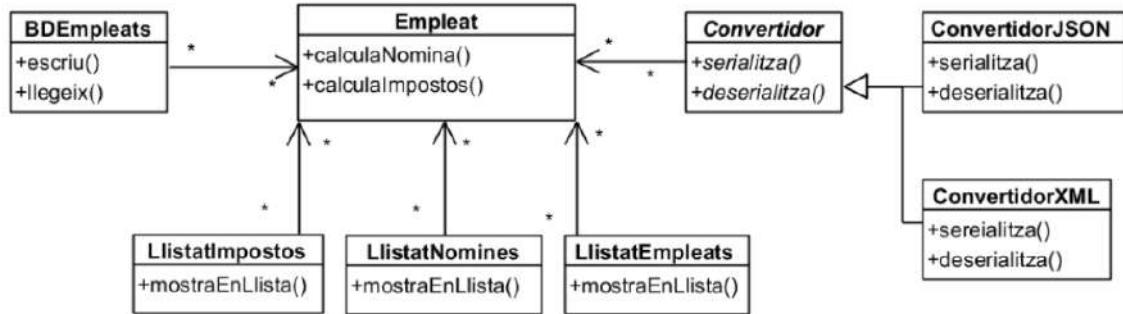
- Està clar què fa
- Fa una cosa única
- És responsable d'aquella cosa

Cohesió - Exemple



- Aquesta classe és un exemple de Cohesió baixa

Cohesió - Exemple

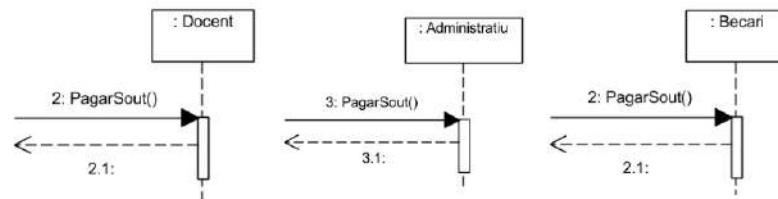
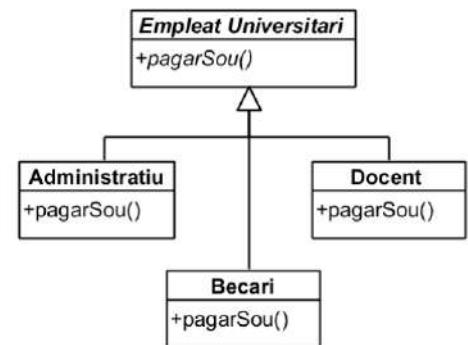


- Aquest sistema és un exemple de cohesió alta

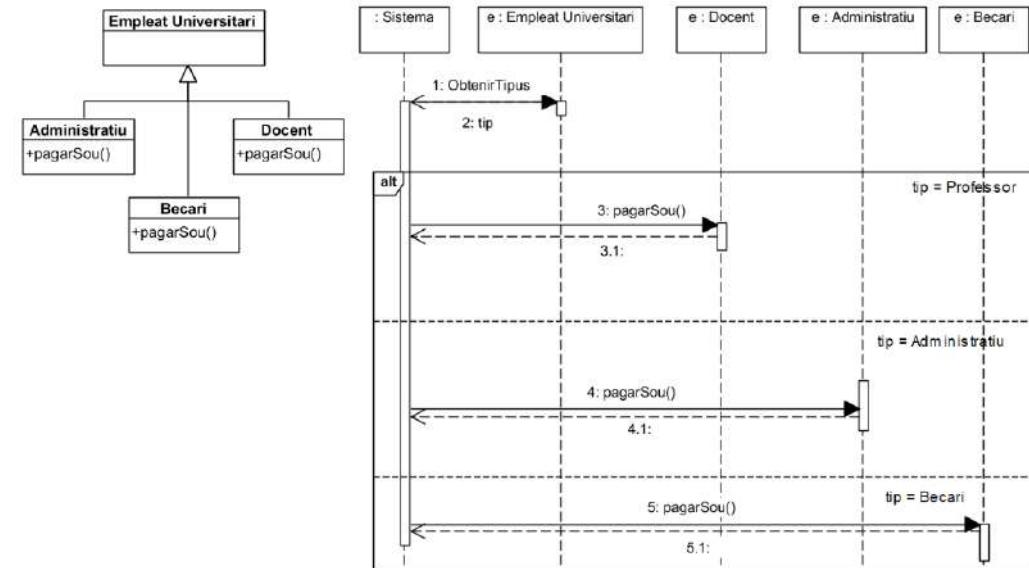
El principi Obert-Tancat (OCP)

- Els mòduls (classes, funcions, etc.) haurien de ser:
 - *Oberts* per a l'extensió. El comportament del mòdul es pot estendre per tal de satisfer nous requisits.
 - *Tancats* per a la modificació. L'extensió no implica canvis en el codi del mòdul. No s'ha de tocar la versió executable del mòdul.
- El comportament dels mòduls que satisfan aquest principi es canvia afegint nou codi, i no pas canviant codi existent.
- Totes les entitats software (classes, mòduls, funcions, etc.) haurien d'estar obertes per extensió, però tancades per modificació

Principi Obert-Tancat - Exemple

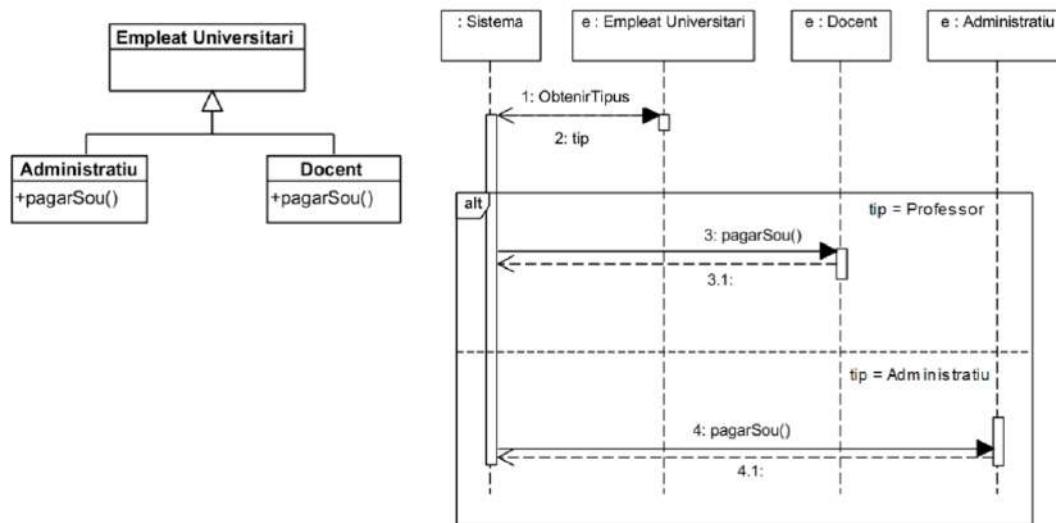


Principi Obert-Tancat - Exemple on no se satisfà



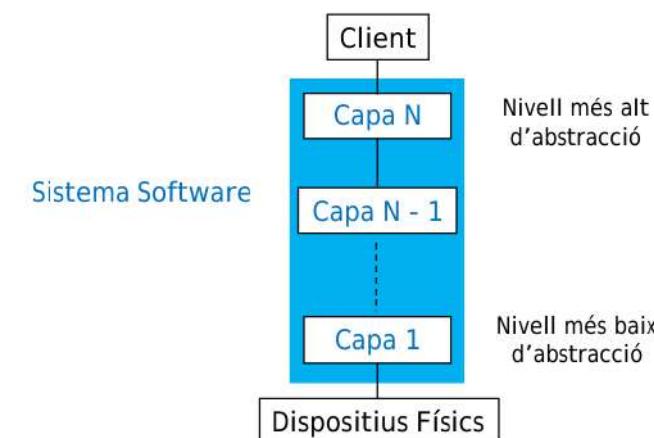
Arquitectura en Capes

Principi Obert-Tancat – Exemple on no se satisfà



Context:

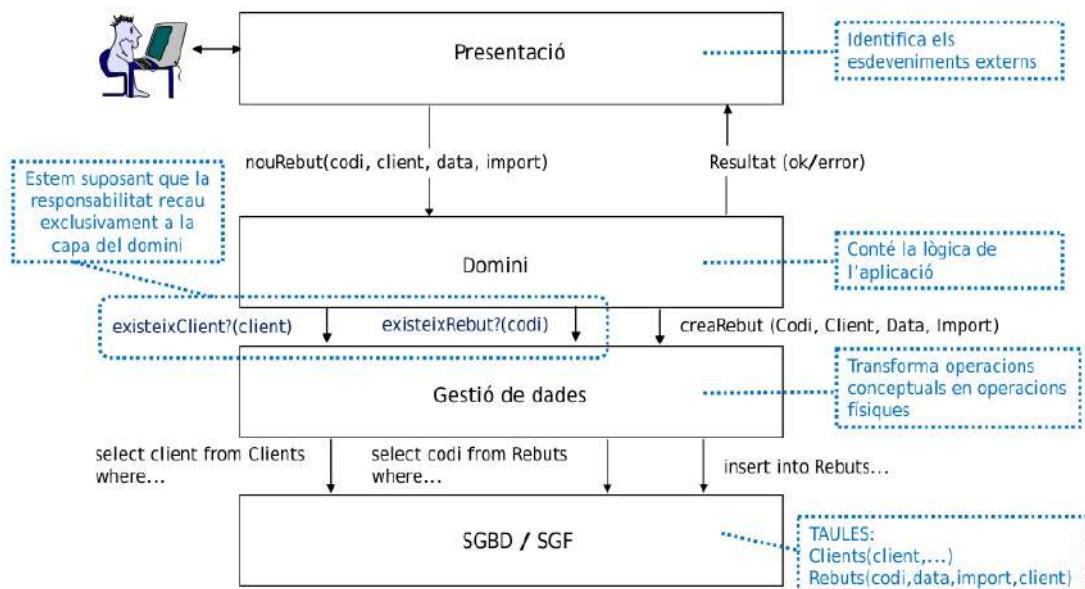
- Un sistema gran que requereix ser descompost en grups de tasques (components), tals que cada grup de tasques està a un nivell determinat d'abstracció.



Arquitectura en 3 capes

- Una de les arquitectures en capes més utilitzada és la arquitectura en 3 capes.
 - Capa de Presentació: Responsable de la interacció amb l'usuari
 - Capa de Domini: Responsable de la implementació del sistema
 - Capa de gestió de Dades: Responsable de la interacció amb el SGBD/F

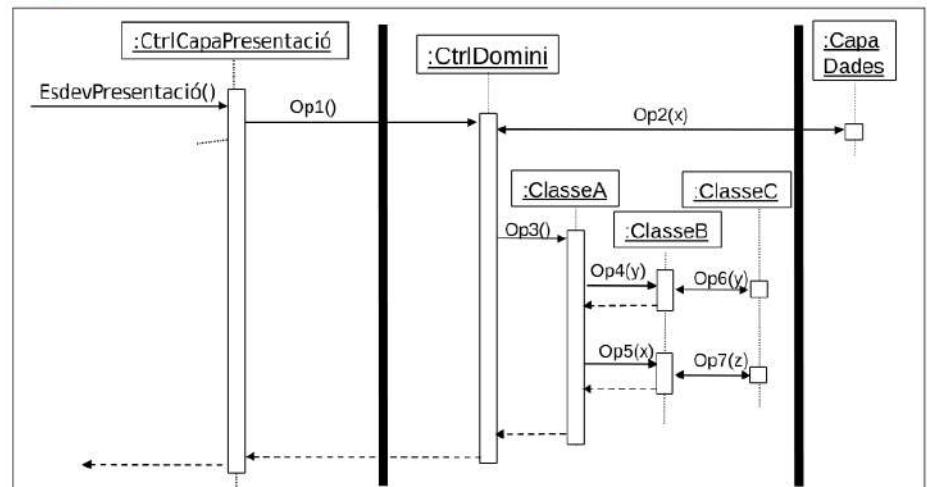
Exemple: comunicació entre capes



Patró Domain Model

- La lògica de l'aplicació resideix bàsicament a la capa del domini
- La capa de domini implementa les seves operacions mitjançant la col·laboració d'instàncies de les seves classes:
 - Ús intensiu del concepte d'assignació de responsabilitats a nivell de classe
- Requereix:
 - Una transformació inicial de l'esquema conceptual d'especificació (dades i operacions) a un diagrama de classes i als contractes de les operacions de disseny
 - Conversió de la classe Data a atribut
- Característiques:
 - (+) Explota la riquesa pròpia de l'orientació a objectes
 - (+) Té a l'abast una col·lecció rica de patrons de disseny
 - (-) Pot no aprofitar-se completament de les funcionalitats ofertes pels SGBD

Domain Model: visió general d'un diagrama de seqüència



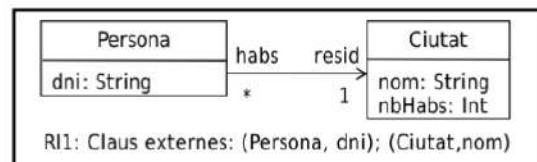
Resultat del disseny a IES:

- Diagrama de classes de disseny
- Contractes de disseny de les operacions
- Diagrama de seqüència de disseny per cada contracte

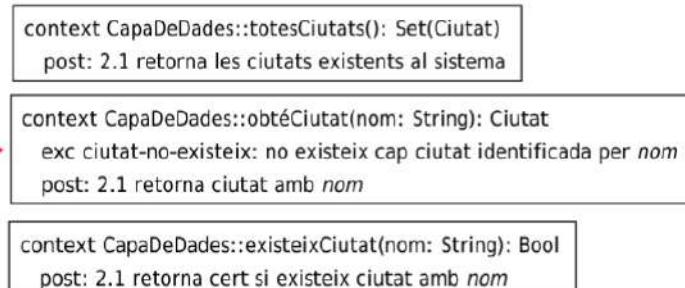
Operacions de la capa de dades

En *Domain Model*, les úniques operacions que contemplen són, per a cada classe, obtenir un objecte donada la seva clau, i obtenir totes les instàncies de la classe

Les actualitzacions a la capa de dades són implícites



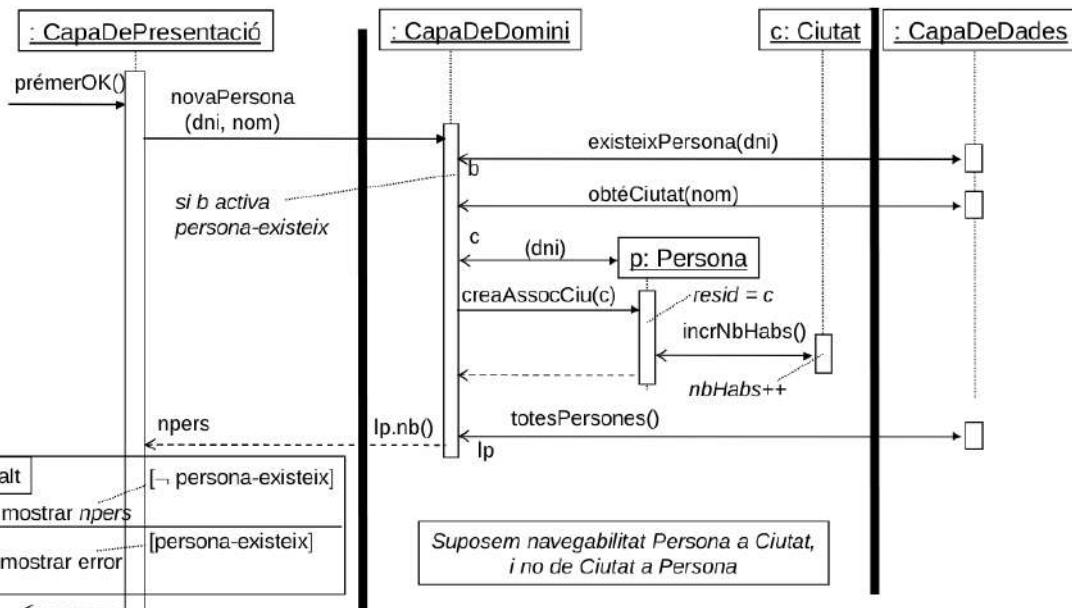
tot i que la controla el SGBD, el mètode és qui comunica l'error



9

Operacions de la capa de dades

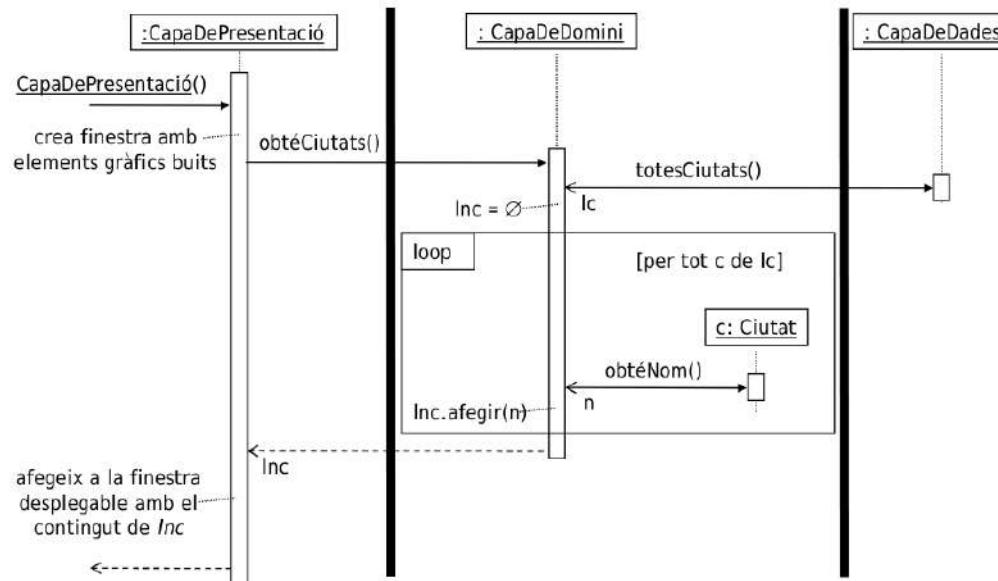
Alta d'informació - exemple



11

Operacions de la capa de dades

Obtenció d'informació - exemple

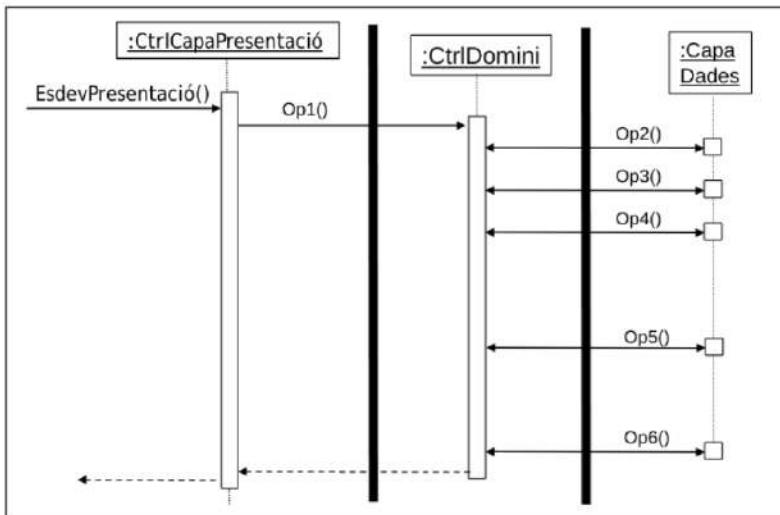


10

Patró Transaction Script

- Procediment que:
 - Rep les dades de la capa de presentació
 - Fa totes les validacions i càlculs necessaris
 - Es comunica amb la capa de dades per consultar i actualitzar la BD
 - Comunica els resultats a la capa de presentació
- Bàsicament, doncs, tenim un procediment per cada transacció de negoci
- La interacció amb la base de dades és totalment explícita
 - El disseny del software es fa considerant el SGBD que s'utilitzarà a la implementació
 - Serà diferent segons usem un SGBD orientat a objectes, relacional, etc.
- Característiques:
 - (+) Paradigma fàcil d'entendre pels programadors
 - (+) Capa de dades molt simple
 - (-) Solució complexa quan la lògica del domini creix
 - (-) La gestió de la persistència és explícita

Transaction Script: visió general d'un diagrama de seqüència



Patró controlador: descripció general

Context:

- Els (sub)sistemes software reben esdeveniments
 - . Ex: la capa de domini d'un SI rep esdeveniments externs
- Un cop interceptats aquests esdeveniments, algun objecte del sistema ha de rebre'ls i executar les accions corresponents

Problema:

- Quin objecte és el responsable de rebre un esdeveniment?

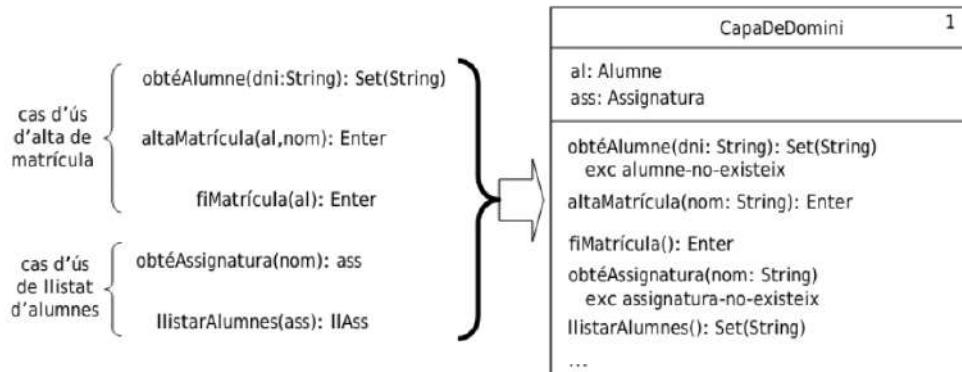
Solució:

- Assignar aquesta responsabilitat a un controlador
 - . Els clients del sistema desconeixen l'estructura interna del sistema
- Un controlador és un objecte d'una certa classe
 - . El controlador delega sobre un o més objectes del sistema el tractament de l'esdeveniment
- L'objecte que tracta l'esdeveniment no té coneixement sobre l'existència o el tipus de controlador
- Variant analitzades:
 - . Façana: Un objecte que representa tot el sistema
 - . Transacció (Command): Un objecte que representa una instància d'esdeveniment

Controlador façana

Aspecte estàtic

- Classe *singleton*
 - tantes operacions com esdeveniments ha de capturar el sistema
 - eventualment, poden incloure's atributs per compartir informació
- Controladors inflats si hi ha molts esdeveniments → poca cohesió



Controlador transacció

Aspecte estàtic (1)

S'introduceix una classe concreta per cada operació del sistema (transacció)

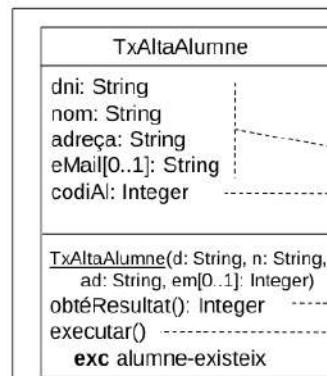
Cada paràmetre de l'operació dóna lloc a un atribut de la classe

- Si l'atribut és *out* o *inout*, s'afegeix una operació per consultar el seu valor
- L'operació constructora de la classe té tants paràmetres com paràmetres *in* i *inout* té l'operació

Si hi ha resultat, també es declara un atribut del tipus del resultat

- S'afegeix una operació per consultar el seu valor

S'afegeix una operació **executar()** que s'encarrega d'executar la transacció



Operació de sistema (transacció):

`altaAlumne(dni: String, nom: String, adreça: String, eMail[0..1]: String): Integer` -- retorna el codi assignat a l'alumne

Paràmetres d'entrada de la transacció

Resultat de la transacció

Operació constructora amb els paràmetres adequats (cal declarar)

Consulta correspondint al resultat de la transacció

S'encarrega d'efectuar les accions corresponents a la transacció

Controlador transacció

Aspecte estàtic (2)

S'introdueix una classe abstracta que actua de superclasse de tots els controladors de transacció del sistema

- Declara l'operació d'executar la transacció com a abstracta
 - Proporciona una vista unificada a les classes clients dels diferents tipus de transaccions

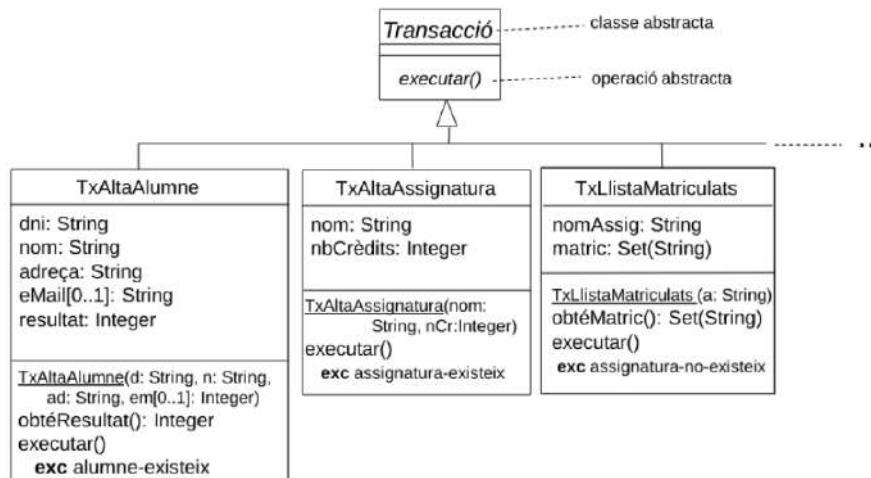
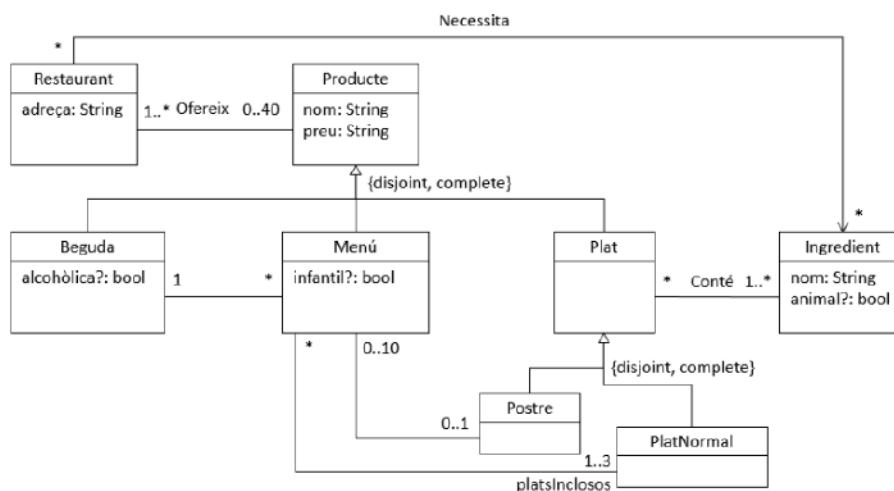


Diagrama de classes de disseny



Restriccions textuais:

1. Claus externes: (Restaurant, adreça), (Producte, nom), (Ingredient, nom)
 2. Un menú infantil no pot tenir una beguda alcohòlica.
 3. Un menú no pot tenir un preu superior a la suma dels preus dels seus productes (beguda, platsInclusos i postre).
 4. Un restaurant ofereix totes les beades que s'inclouen en els seus menús oferts.

Contracte de l'operació IlistaMenusVegetarians

Operació: llistaMenusVegetarians(adreça: String): Set(TupleType(nom: String, preu: int, estalvi: int))

Exc: [NoExisteixRestaurant]: el restaurant identificat per *adreça* no existeix

Body:

Per cada menú ofert en el restaurant identificat per *adreça*, tal que ni cap dels seus plats inclosos ni el postre tenen un ingredient d'origen animal, es retorna la següent informació:

- el nom i el preu del menú
 - L'estalvi. És a dir, la diferència entre el preu del menú, i la suma dels preus dels platsInclusos, de la beguda i del postre.

Contracte de l'operació crearMenú

Operació: crearMenú(adreça: String, nomMenu: String, preu: int, nomPlat: String, nomPostre: String, nomBeguda: String, infantil: bool)

Excel

- [NoExisteixRestaurant]: el restaurant identificat per *adreça* no existeix.
 - [NoExisteixPlatNormal]: El plat normal identificat per *nomPlat* no existeix.
 - [NoExisteixPostre]: El postre identificat per *nomPostre* no existeix.
 - [NoExisteixBeguda]: La beguda identificada per *nomBeguda* no existeix.
 - [JaExisteixProducte]: Ja existeix un producte amb nom *nomMenu*.
 - [MassaProductes]: El restaurant *adreça* ja ofereix 40 productes.
 - [PostreEnMassaMenús]: El Postre *nomPostre* ja s'usa en 10 Menús.
 - [MenúInfantilIncorrecte]: *infantil* val cert quan la beguda *nomBeguda* és alcohòlica.
 - [RestaurantNoOfereixBeguda]: El restaurant *adreça* no ofereix la beguda *nomBeguda*
 - [PreuIncorrecte]: El *preu* del menú és superior a la suma del preu del *nomPlat*, *nomPostre* i *nomBeguda*.

Post:

- Es dóna d'alta un nou Menú amb els valors de nom, preu i infantil passats per paràmetre, el plat *nomPlat* com a únic platInclós, la beguda *nomBeguda*, i el postre *nomPostre*.
 - S'enregistra que el Restaurant *adreça* ofereix aquest nou Menú.
 - S'actualitzen els ingredients necessitarats pel Restaurant *adreça*.

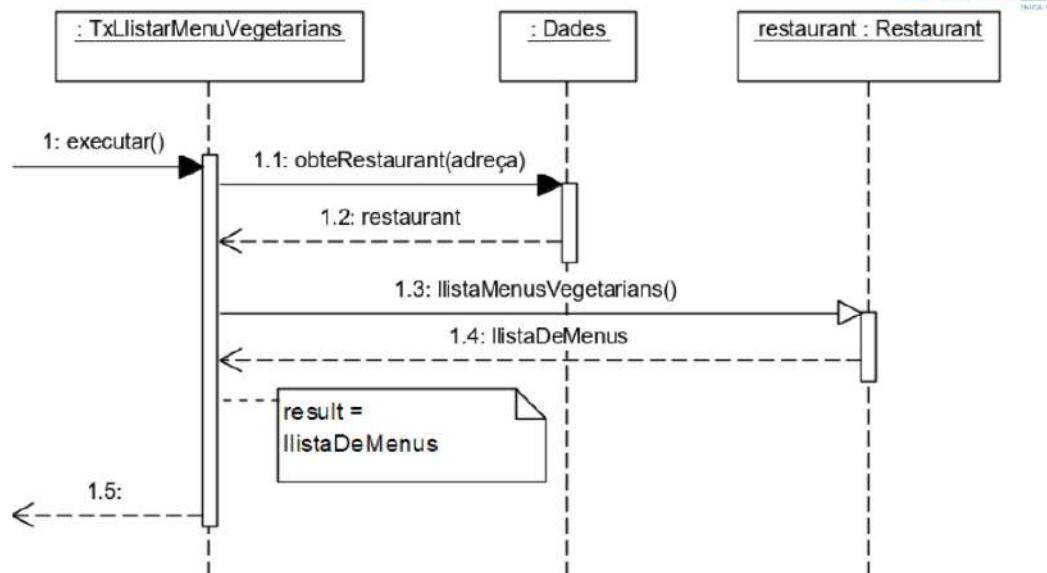
LlistaMenusVegetarians

Restaurant::LlistaMenusVegetarians

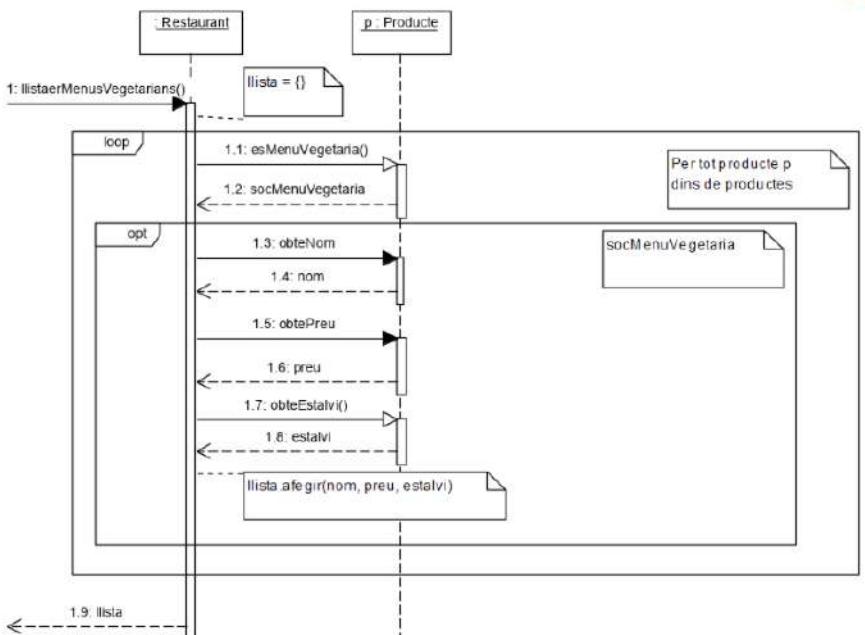
- Creem una classe que segueix el *Controlador Transacció* que conté
 - Un atribut per cada paràmetre de la operació
 - Un atribut resultat del tipus retornat per la operació
 - Una operació *Executar()*
 - Una operació de *ObteResultat()*



Executar

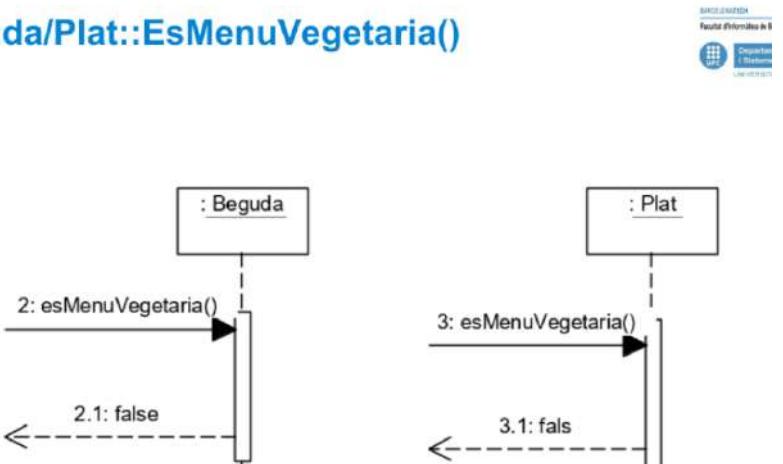


- Obtenim el restaurant de la Capa de Dades.
 - Això pot llençar la Excepció [NoExisteixRestaurant]
- Deleguem la resta de la operació a la classe Restaurant
- Ens guardem el resultat



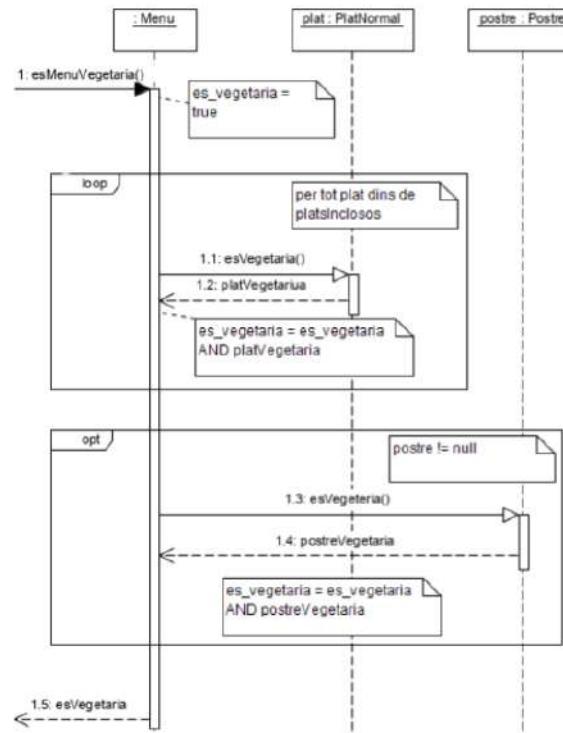
- Per cada Producte Ofert, mirem si es tracta d'un Menú Vegetarià amb una operació Abstracta
- Si és un Menú Vegetarià, guardem la seva informació a "llista", però necessitarem una nova operació abstracta: ObtéEstalvi()

Beguda/Plat::EsMenuVegetaria()



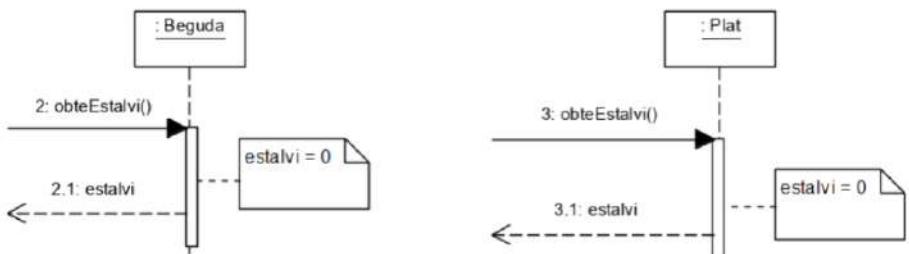
- Beguda i Plat implementen EsMenuVegetarià retornant "fals", ja que cap dels dos és un Menú

Menu::EsMenuVegetarià



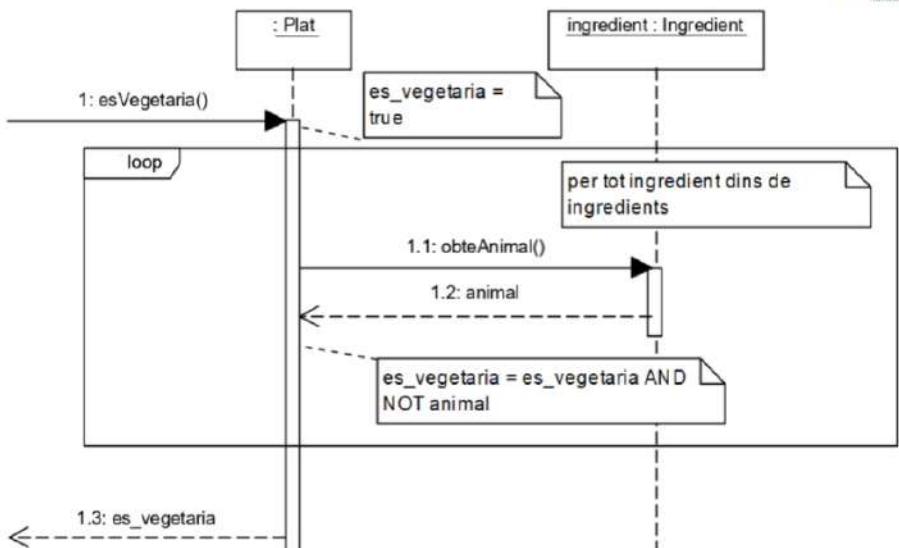
- Iterem per tots els platsInclosos i mirem si tots són vegetarians
- Mirem si el postre és vegetarià també
- L'operació nova `EsVegetarià` és comuna a `Postre` i `PlatNormal`, així que la posarem a `Plat`, per tal de reaprofitar-la en els dos casos

Beguda/Plat::ObteEstalvi()



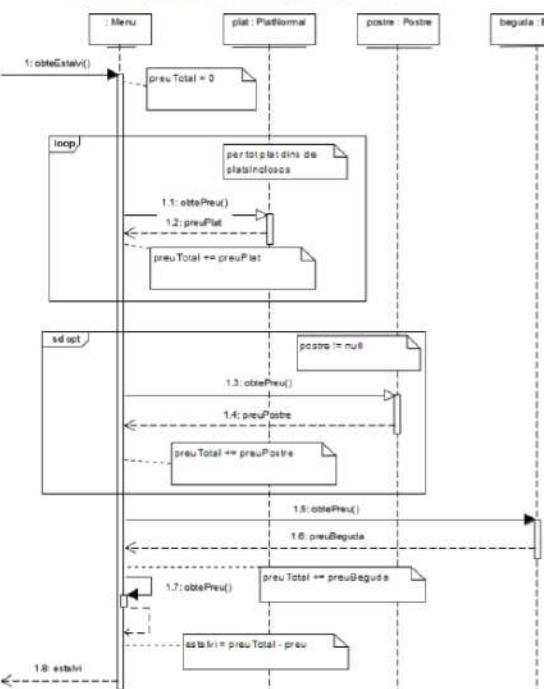
- L'Estalvi és un concepte que només aplica als Menus, així que Plats i Begudes tenen un estalvi de 0

Plat::EsVegetaria()



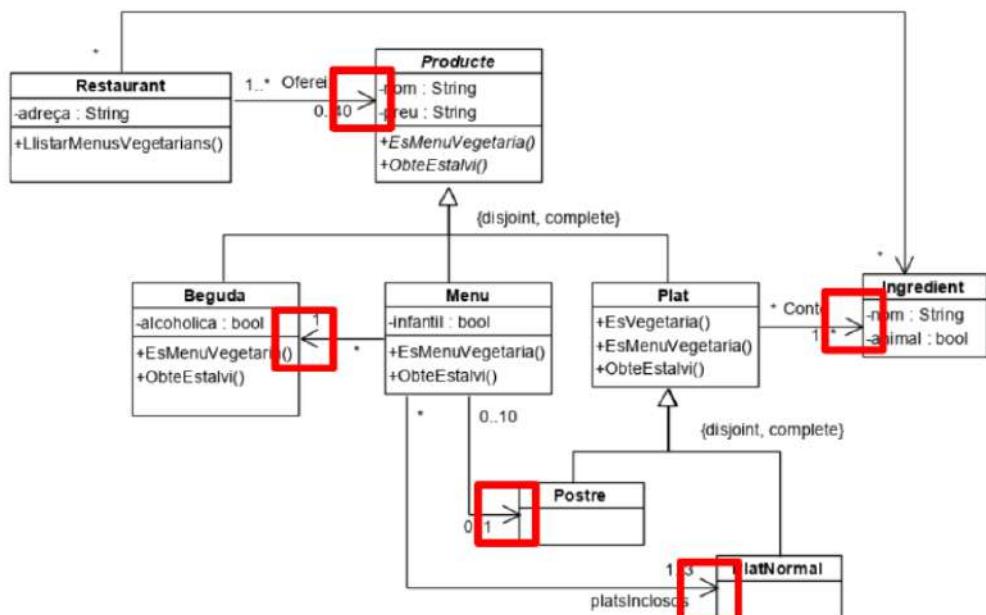
- Mirem per tots els ingredients i garantim que cap no és d'origen animal

Menu::ObteEstalvi



- Obtenim el Preu de tots els platsInclosos i el sumem
- Sumem també el preu del postre
- Sumem també el preu de la beguda
- L'Estalvi serà el preu sumat fins ara menys el preu del menú

Noves Navegabilitats i Operacions

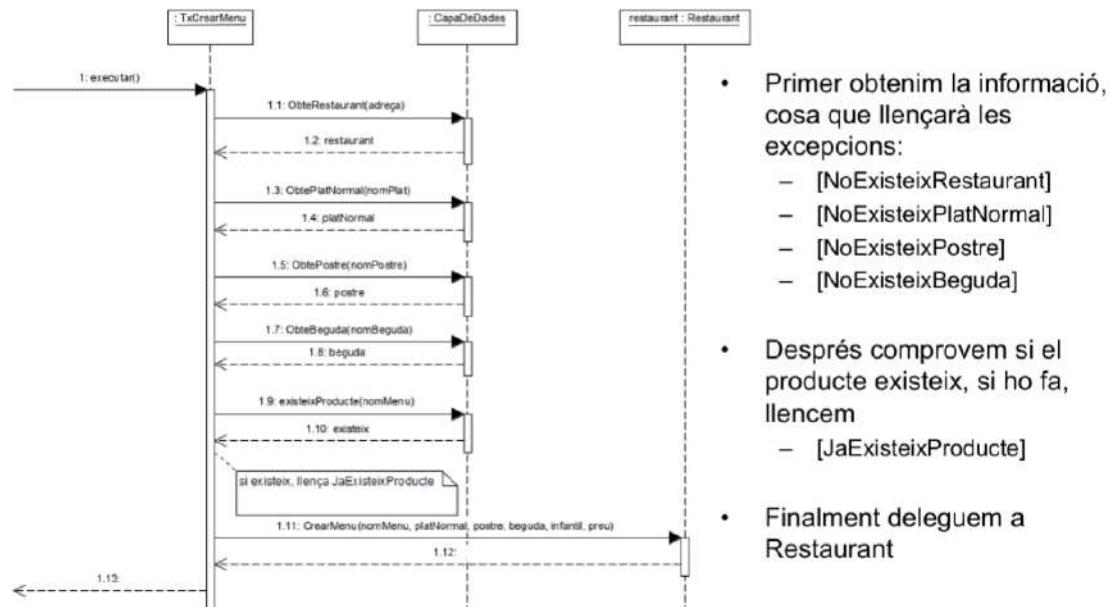


CrearMenu

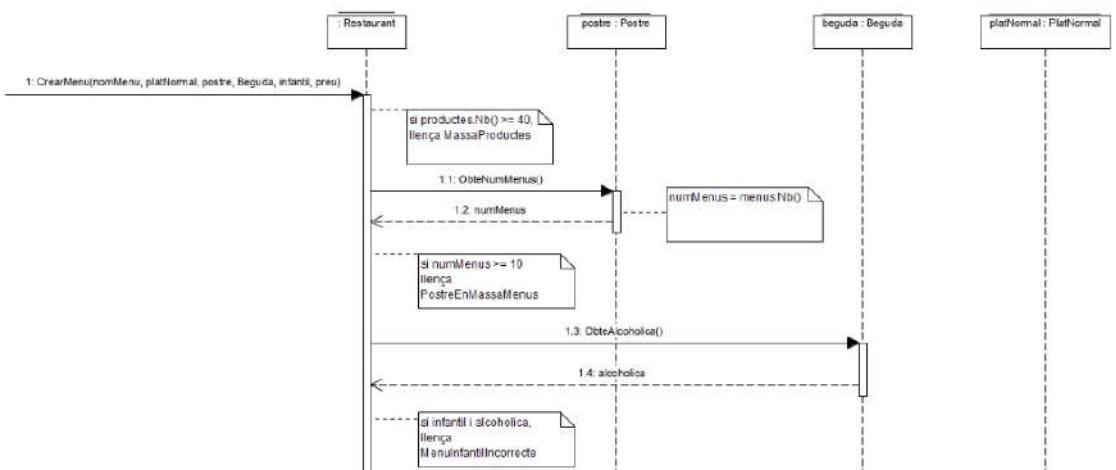
txCreateMenu	
-adreça : String	
-nomMenu : String	
-preu : int	
-nomPlat : String	
-nomPostre : String	
-nomBeguda : String	
-infantil : bool	
+executar()	

- Creem una classe que segueix el *Controlador Transacció* que conté
 - Un atribut per cada paràmetre de la operació
 - Una operació *Executar()*
- En aquest cas no cal resultat perquè l'operació no retorna res

Executar

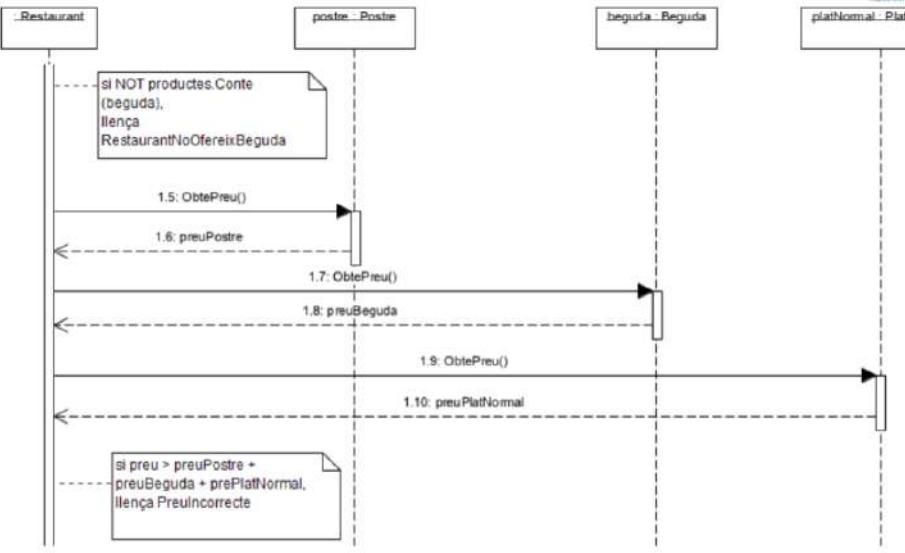


Menu::CrearMenu (Part 1)



- Fem les comprovacions de
 - [MassaProductes]
 - [PostreEnMassaMenús]
 - [MenúInfantilIncorrecte]
- Necessitarem una nova operació a Postre: *ObteNumMenus*

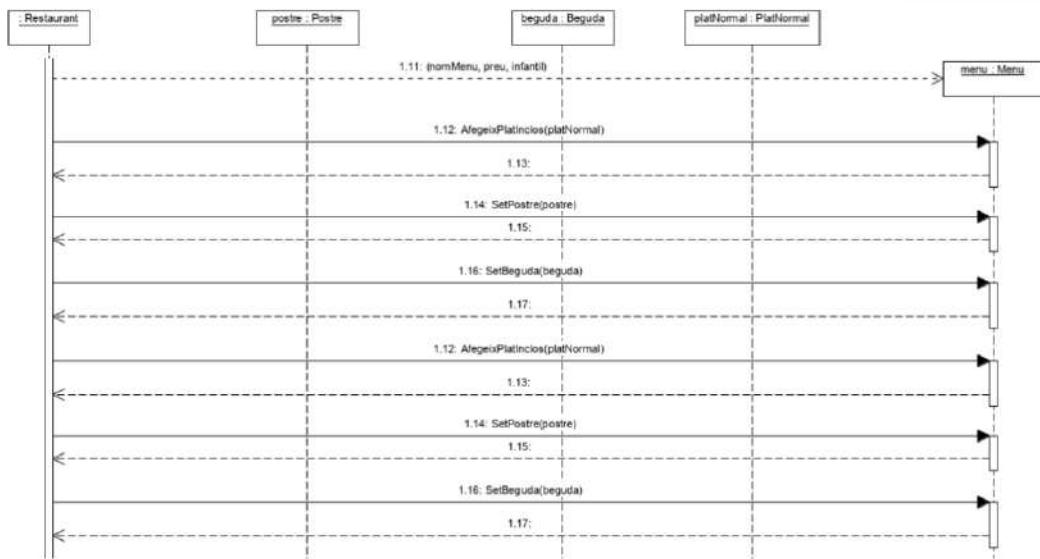
Menu::CrearMenu (Part 2)



- Comprovem les excepcions:
 - [RestaurantNoOfereixBeguda]
 - [PreulIncorrecte]

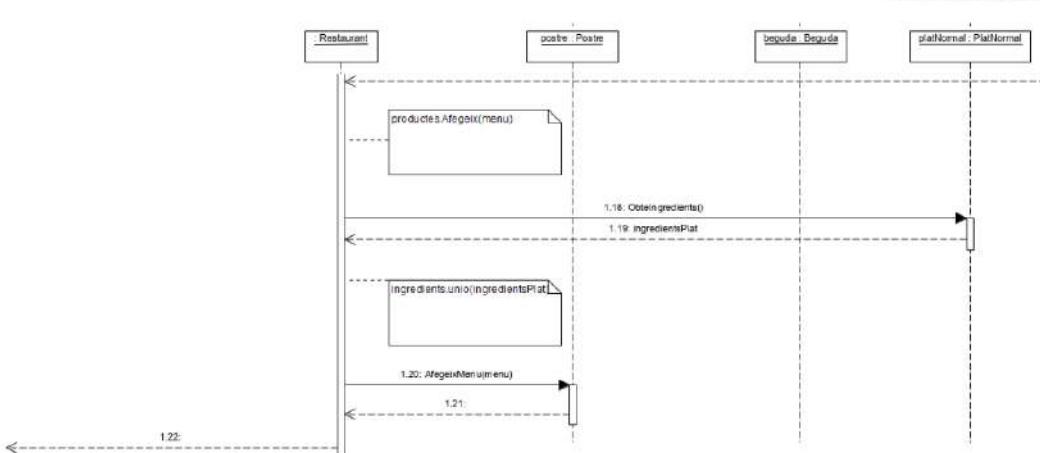
- Ara ja podem crear l'objecte nou

Menu::CrearMenu (Part 3)



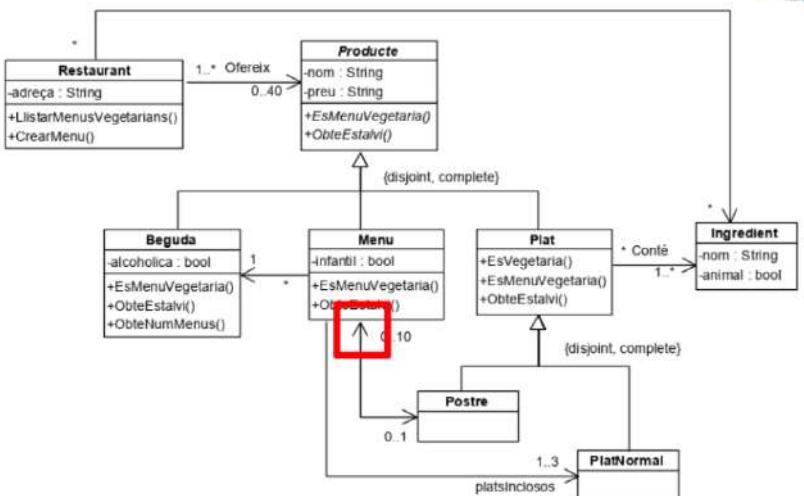
- Creem el nou Menu i el relacionem amb totes les instàncies que ens indiquen les navegabilitats

Menu::CrearMenu (Part 4)



- Relacionem el Menú amb aquelles instàncies que tenen navegabilitat cap a Menú
 - Restaurant (via produc)
 - Postre
- Actualitzem la llista d'Ingredients
- Retornem

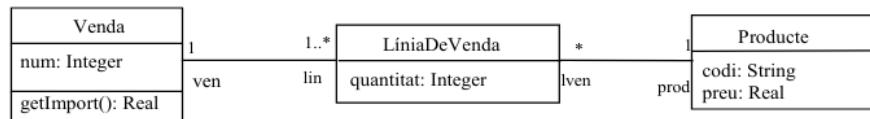
Alerta! Nova navegabilitat!



- Abans de Crear la instància nova, cal fer una repassada a possibles navegabilitats noves afegides.
- Hem afegit una nova navegabilitat entre Postre i Menu que s'ha de tenir en compte.

Exercici 1

Donat el diagrama de classes de disseny següent:

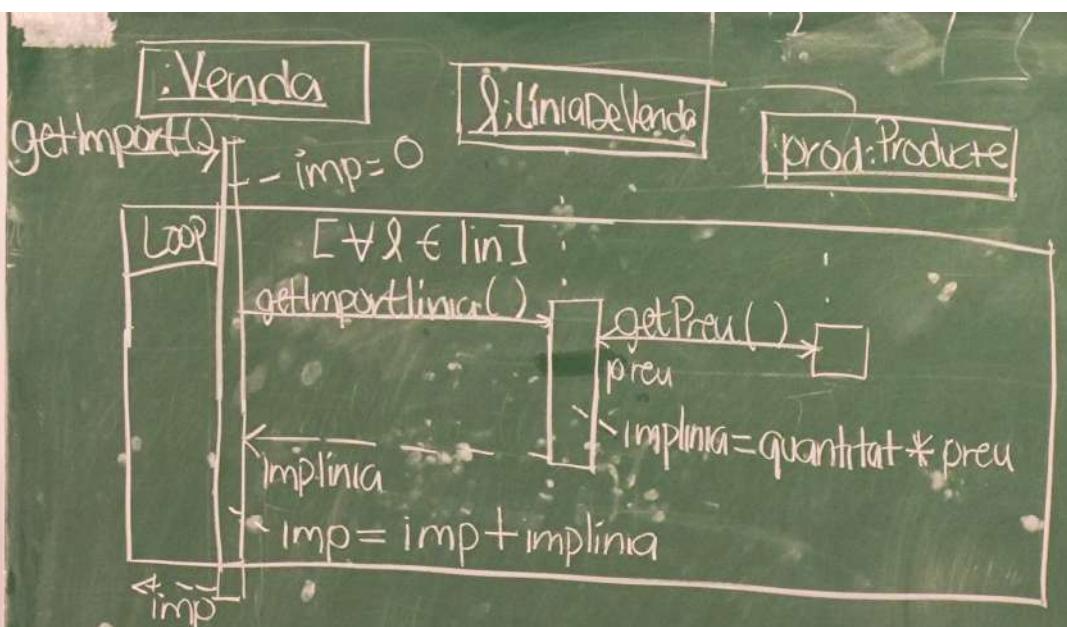


i donat el contracte següent:

context Venda::getImport():Real

post: result= suma de quantitat * preu del producte de totes les línies de la venda

Fer el diagrama de seqüència de l'operació *getImport():Real* i de totes les operacions que siguin invocades en aquesta operació. Poseu comentaris de tot el que no hi aparegui de forma explícita. Cal que indiqueu al diagrama de classes donat la navegabilitat resultant del vostre disseny.

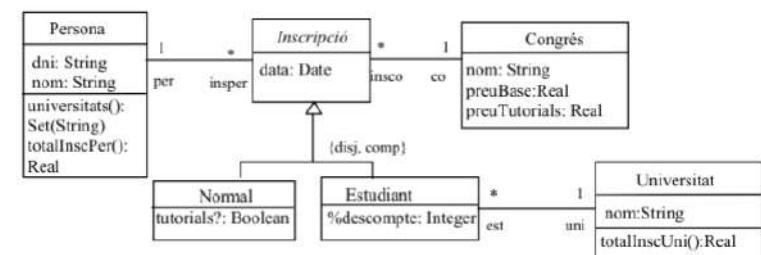


Navegabilitat:

Venda → LíniaDeVenda
LíniaDeVenda → Producte

Exercici 2

Donat el diagrama de classes de disseny següent:

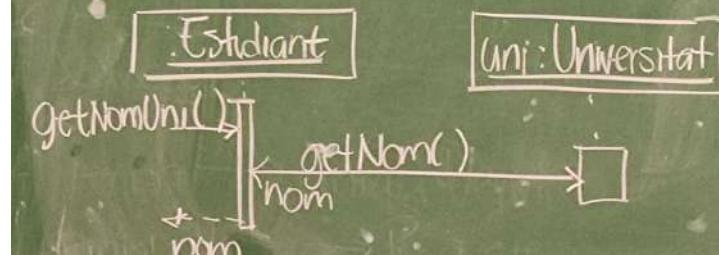
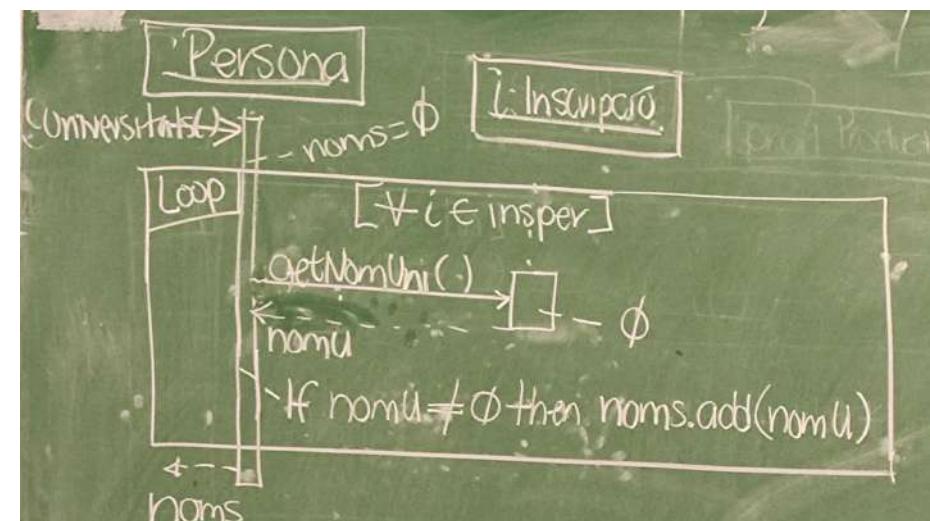


i donat el contracte següent:

context Persona::universitats():Set(String)

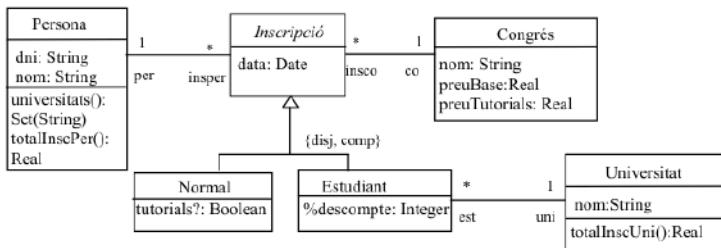
post: result= tots els noms de les universitats per les quals la persona té alguna inscripció com a estudiant

Fer el diagrama de seqüència de l'operació *universitats():Set(String)* i de totes les operacions que siguin invocades en aquesta operació. Poseu comentaris de tot el que no hi aparegui de forma explícita. Cal que indiqueu al diagrama de classes donat la navegabilitat resultant del vostre disseny.



Exercici 3

Donat el diagrama de classes de disseny següent:



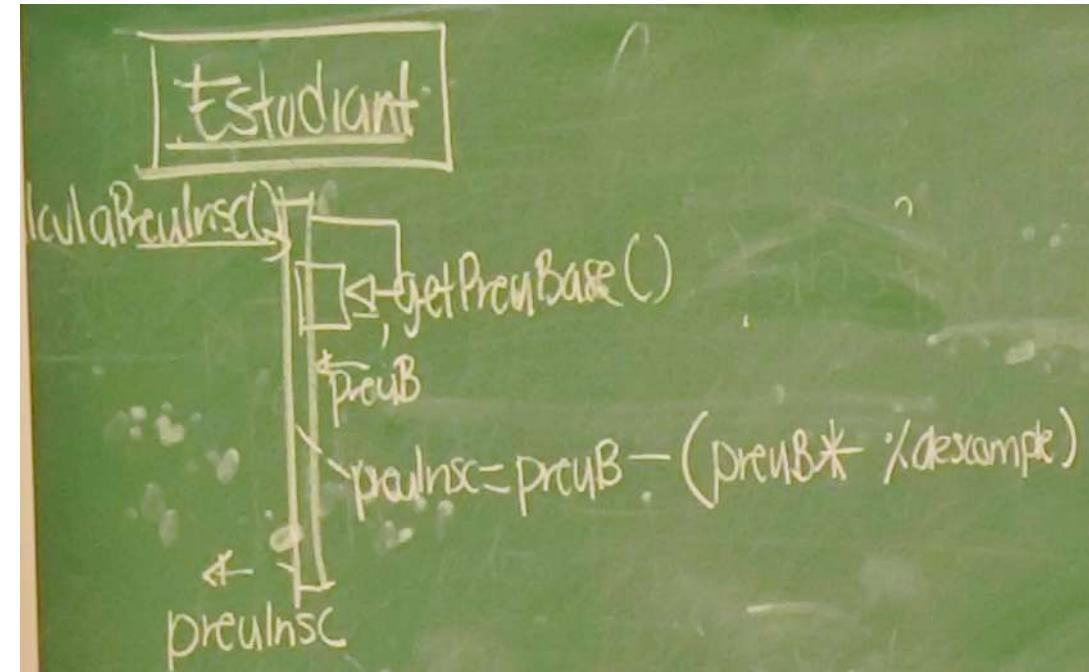
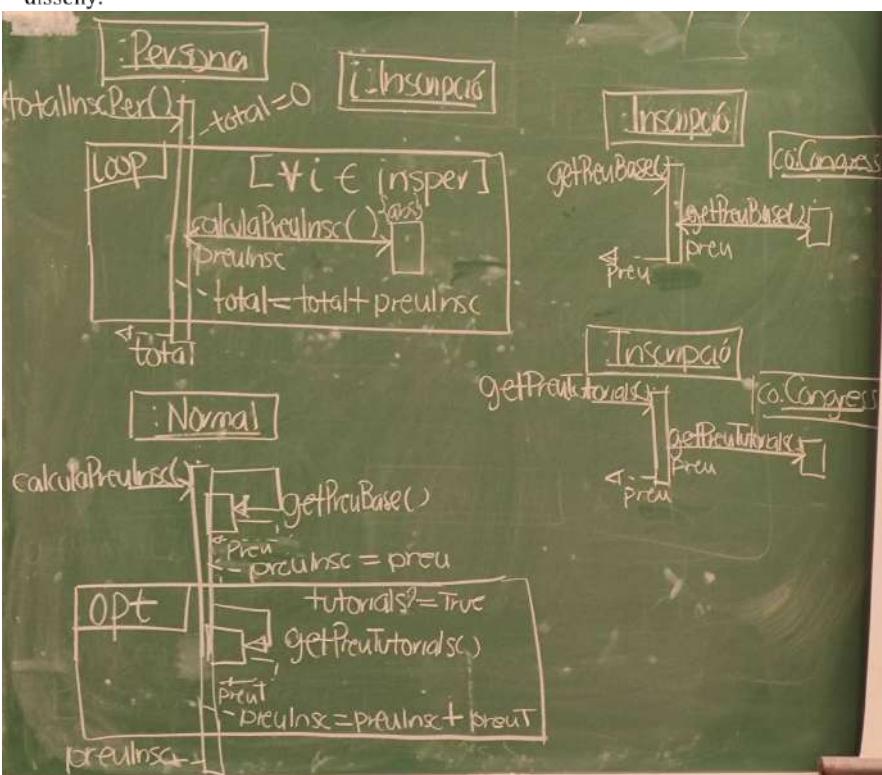
i donat el contracte següent:

context Persona::totalInscPer():Real

post: result= suma dels imports de totes les inscripcions de la persona. L'import d'una inscripció es calcula com segueix:

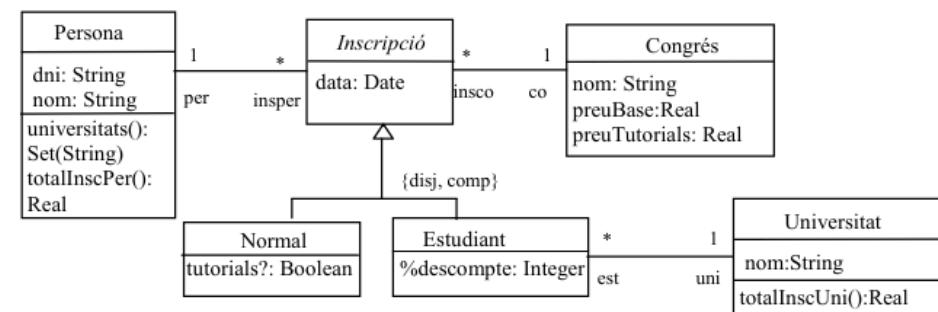
- si la inscripció és de tipus estudiant l'import és el preu base del congrés menys el descompte segons el %descompte que tingui
- si la inscripció és de tipus normal l'import és el preu base del congrés més el preu dels tutorials si tutorials? és cert.

Fer el diagrama de seqüència de l'operació `totalInscPer():Real` i de totes les operacions que siguin invocades en aquesta operació. Poseu comentaris de tot el que no hi aparegui de forma explícita. Cal que indiqueu al diagrama de classes donat la navegabilitat resultant del vostre disseny.



Exercici 4

Donat el diagrama de classes de disseny següent:

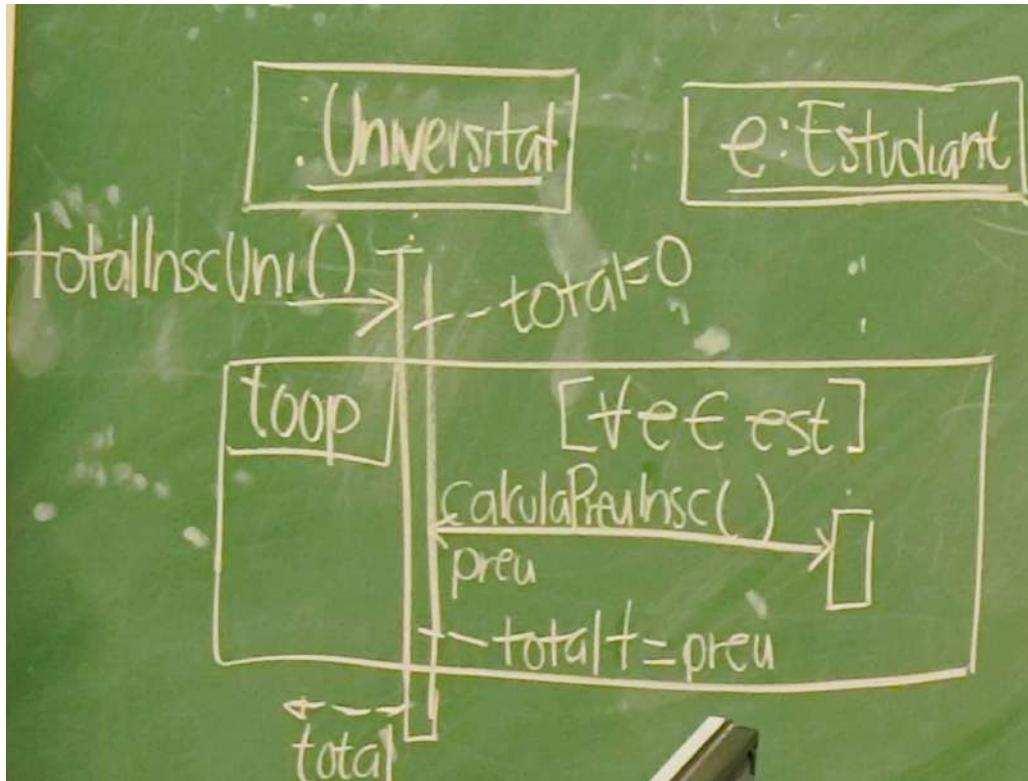


i donat el contracte següent:

context Universitat::totalInscUni():Real

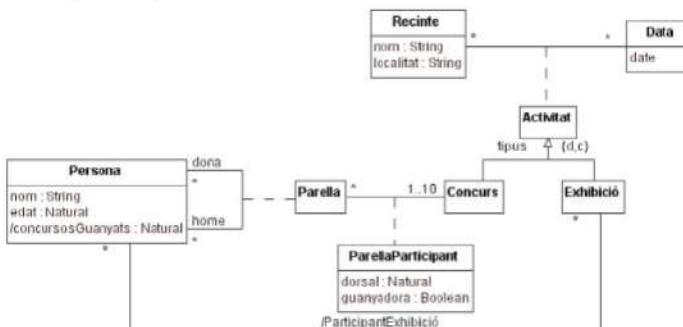
post: result= suma de l'import de totes les inscripcions de tipus estudiant de la universitat.

Fer el diagrama de seqüència de l'operació `totalInscUni():Real` i de totes les operacions que siguin invocades en aquesta operació. Poseu comentaris de tot el que no hi aparegui de forma explícita. Cal que indiqueu al diagrama de classes donat la navegabilitat resultant del vostre disseny.



Una escola de ball organitza dos tipus d'activitat, concursos i exhibicions de ball. Les activitats s'organitzen en una data i un recinte determinat, del que en sabem el nom que l'identifica i la localitat on es troba. El sistema també emmagatzema el nom i l'edat de les persones que, per parelles, participen a un concurs, i se'ls hi assigna un número de dorsal al concurs. De cada concurs també s'enregistren les parelles guanyadores. A les exhibicions hi participen totes les personnes que hagin estat guanyadores en algun concurs de l'any anterior a l'any de la celebració de l'exhibició. A continuació disposeu de l'especificació feta per aquest sistema.

Esquema conceptual d'especificació:



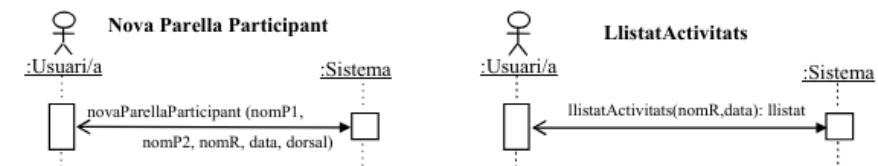
Restriccions textuais

1. Claus externes: (Persona, nom), (Recinte, nom), (Data, data)
2. Els membres d'una parella han de ser diferents
3. Una persona no pot participar més d'una vegada al mateix concurs (com a membre de parelles diferents)
4. En un concurs no hi ha dorsals repetits
5. En un mateix dia no hi pot haver dues exhibicions

Informació derivada

1. /concursosGuanyats d'una Persona és el nombre de concursos en que la persona ha format part d'una parella guanyadora
2. Les persones que participen a una exhibició són tots els que han estat guanyadors d'alguns concurs de l'any anterior a l'any de l'exhibició

Diagrama de seqüència d'esdeveniments del sistema:



Contracte de l'operació novaParellaParticipant:

Operació: novaParellaParticipant (nomP1: String, nomP2: String, nomR: String, data: Date, dorsal: Integer)

Pre:

- Existeix la parella identificada per *nomP1* i *nomP2*.
- Existeix el concurs identificat per *nomR* i *data*.

Post:

- Es dóna d'alta una instància de ParellaParticipant definida per la parella (*nomP1*, *nomP2*) i pel concurs (*nomR*, *data*), amb el dorsal *dorsal* i l'atribut guanyadora a fals.

Contracte de l'operació IlistatActivitats:

Operació: IlistatActivitats (nomR: String, data: Date): Set(String)

Pre:

- L'activitat identificada per *nomR* i *data* existeix.

Body:

- Si l'activitat és una exhibició, es mostren els noms de totes persones que hi ha participat.
- Si l'activitat és un concurs, es mostren els noms de les dones de les parelles participants al concurs que no varen ser parelles guanyadores.

Tenint en compte que:

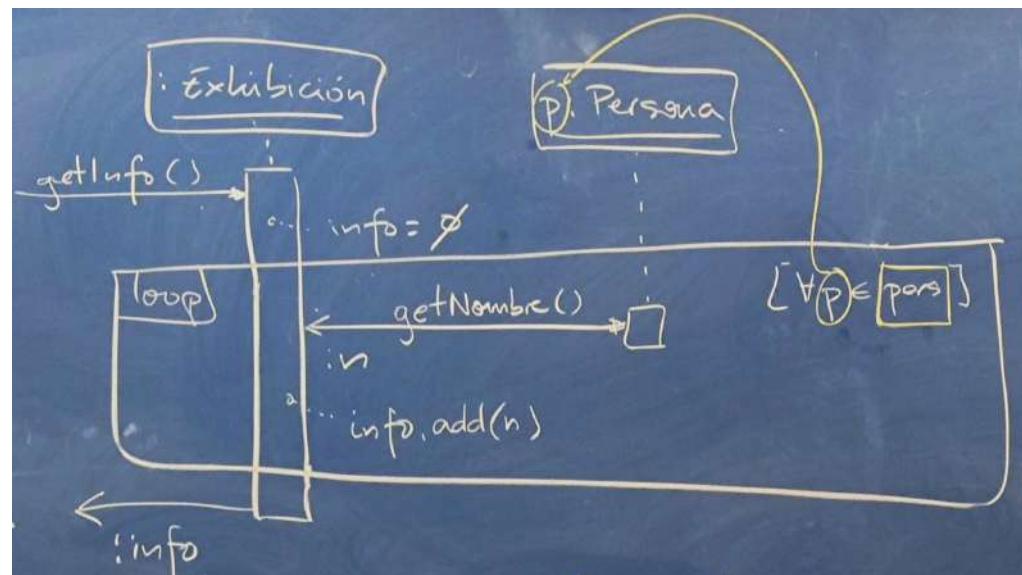
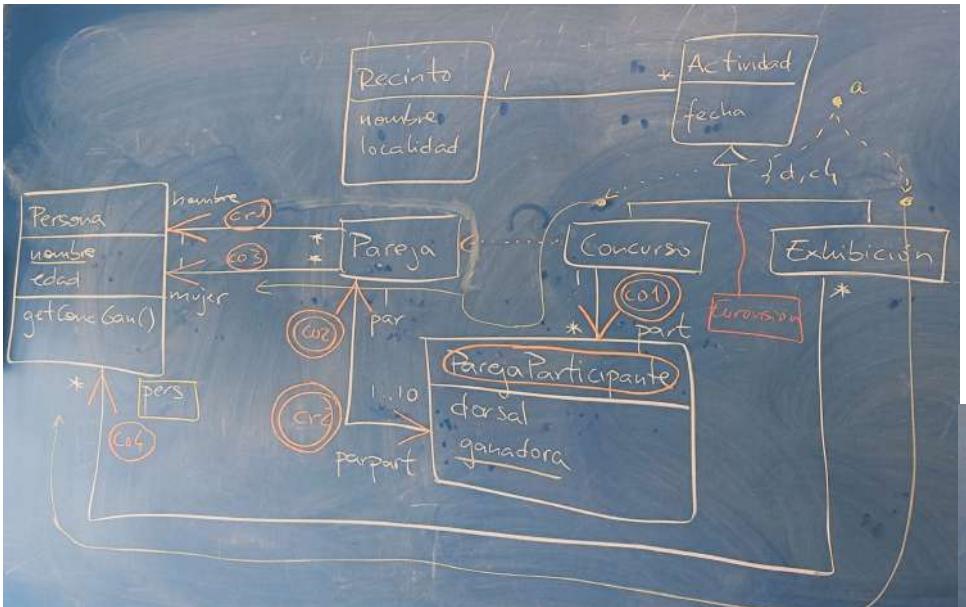
- l'associació *participantExhibició* ha de ser materialitzada.
- l'atribut *concursosGuanyats* ha de ser calculat
- cal utilitzar el controlador transacció

Es demana:

- a) Diagrama de classes de disseny obtingut a partir de l'esquema conceptual de les dades, indicant explícitament les restriccions d'integritat que apareixen o desapareixen, i els contracte de les operacions obtinguts com a conseqüència de la traducció de l'esquema d'especificació al de disseny.

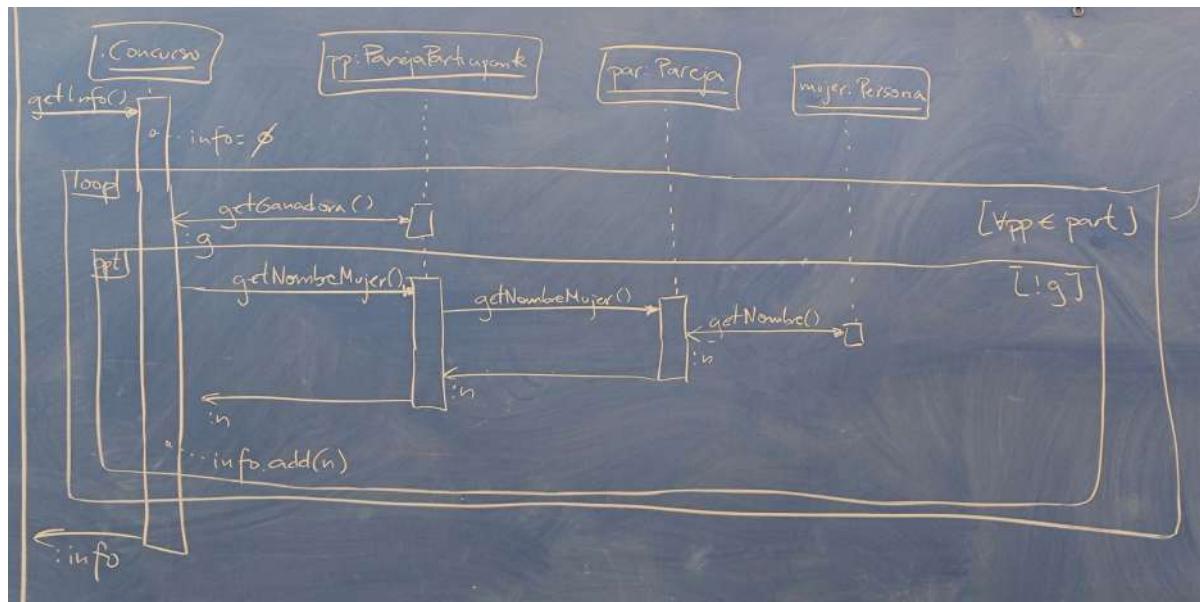
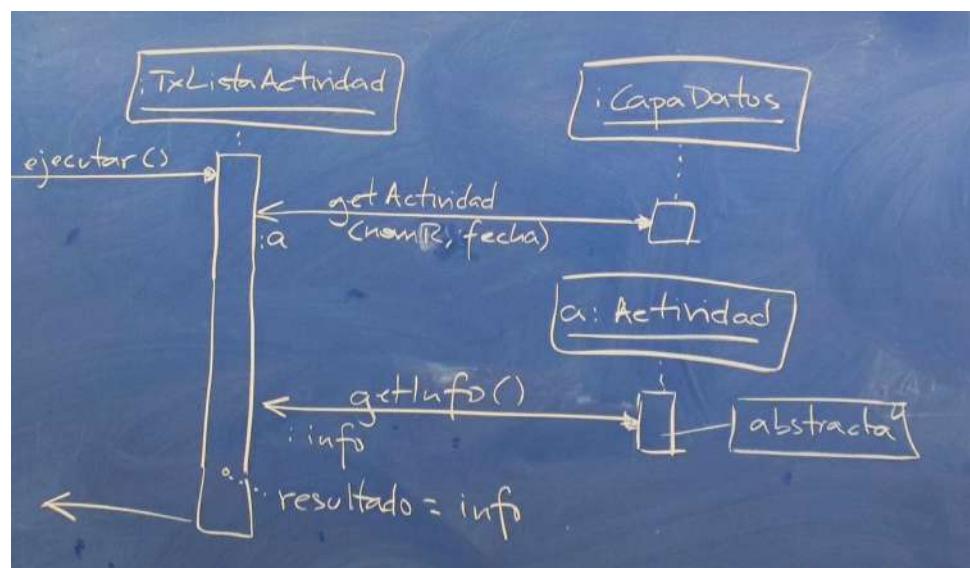
b) Diagrama de seqüència de les operacions `novaParellaParticipant` i `llistatActivitats` i de totes les operacions que siguin invocades en aquest diagrama de seqüència. Poseu comentaris de tot el que no hi aparegui de forma explícita. Especifiqueu també quines de les operacions que heu utilitzat al vostre disseny són abstractes.

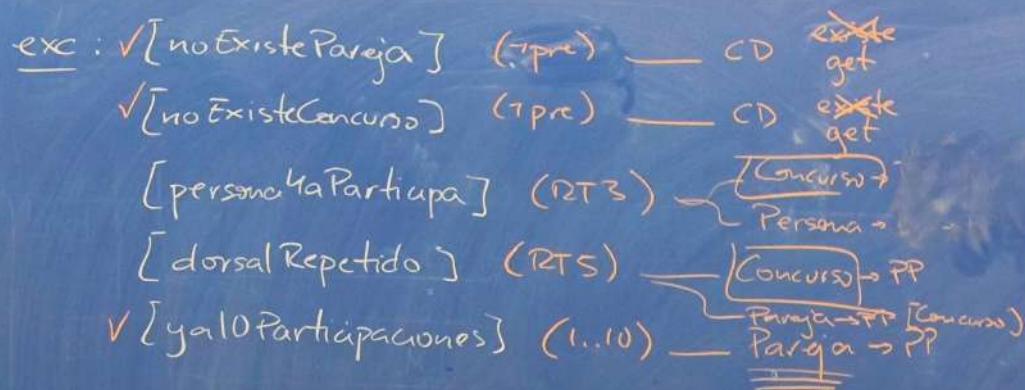
c) Indiqueu al diagrama de classes de l'apartat a) la navegabilitat resultant del vostre disseny.



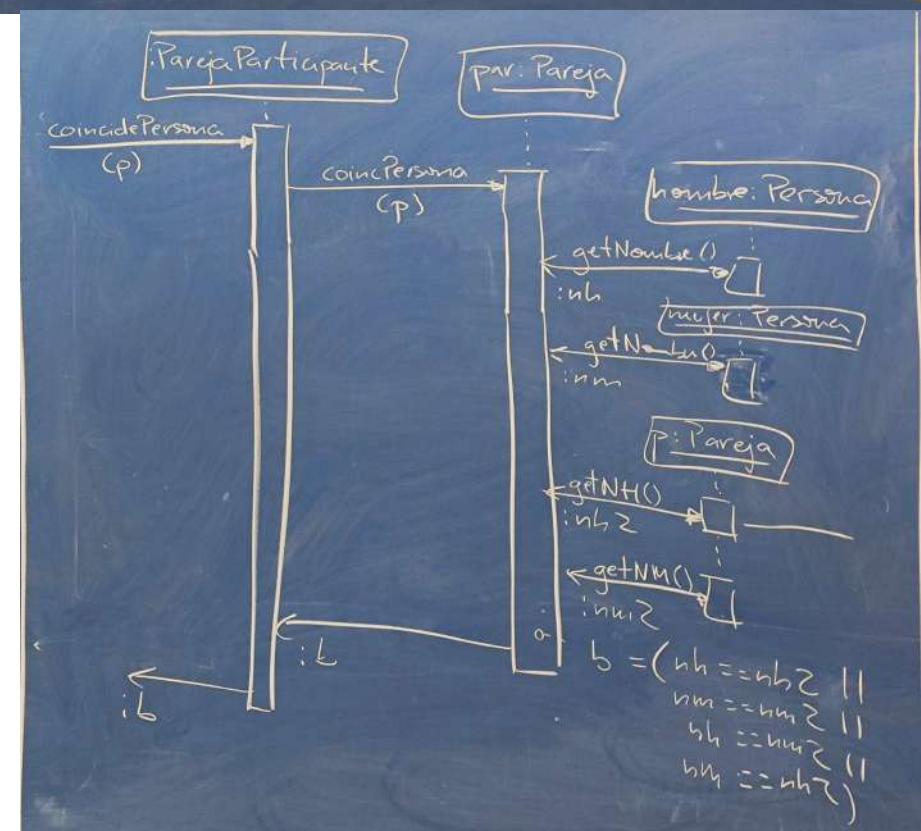
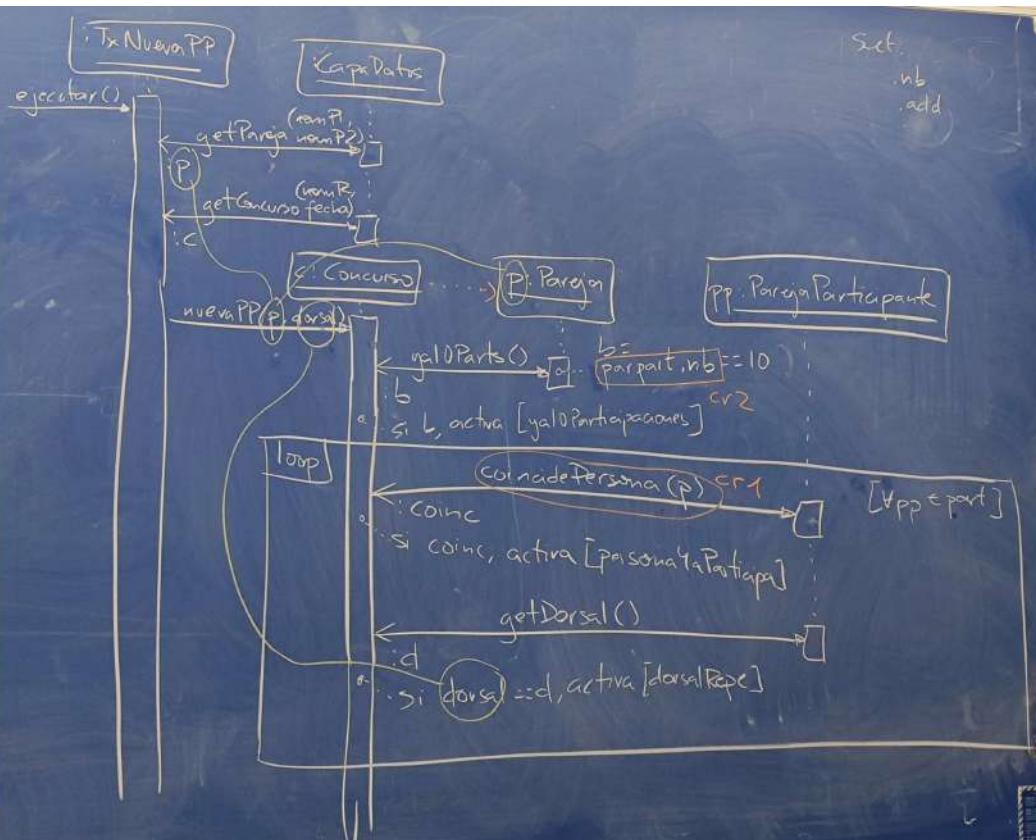
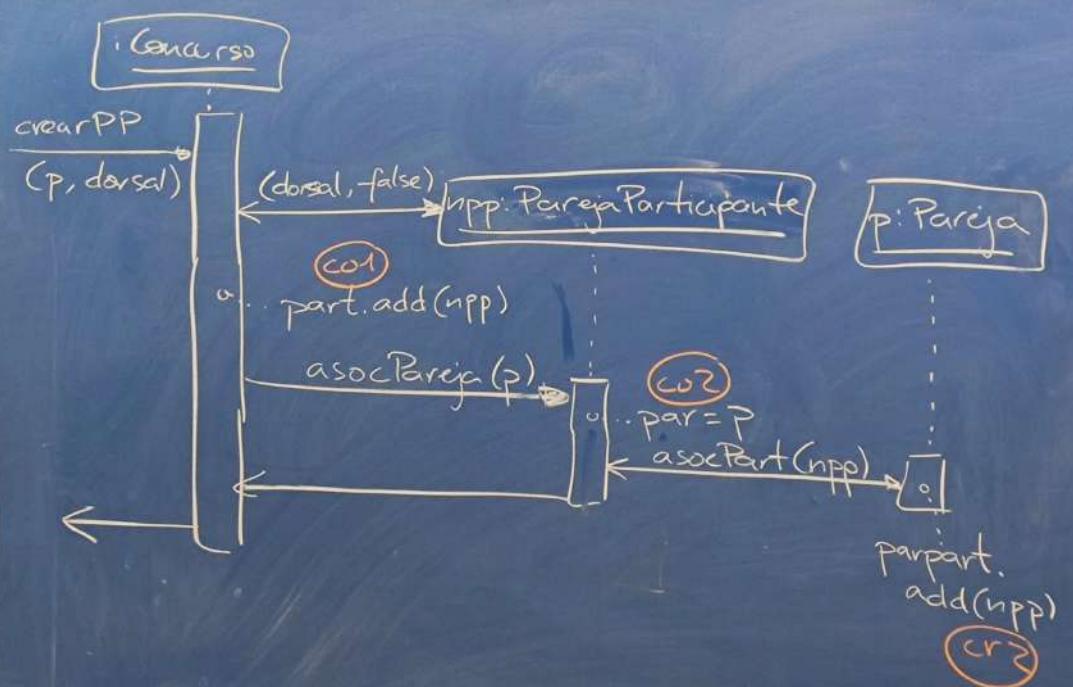
RT añadidas:

- RT6. 82 actividades
mismo recinto
y fecha
- RT7. 82 parejas
con mismo
hombre y
mujer



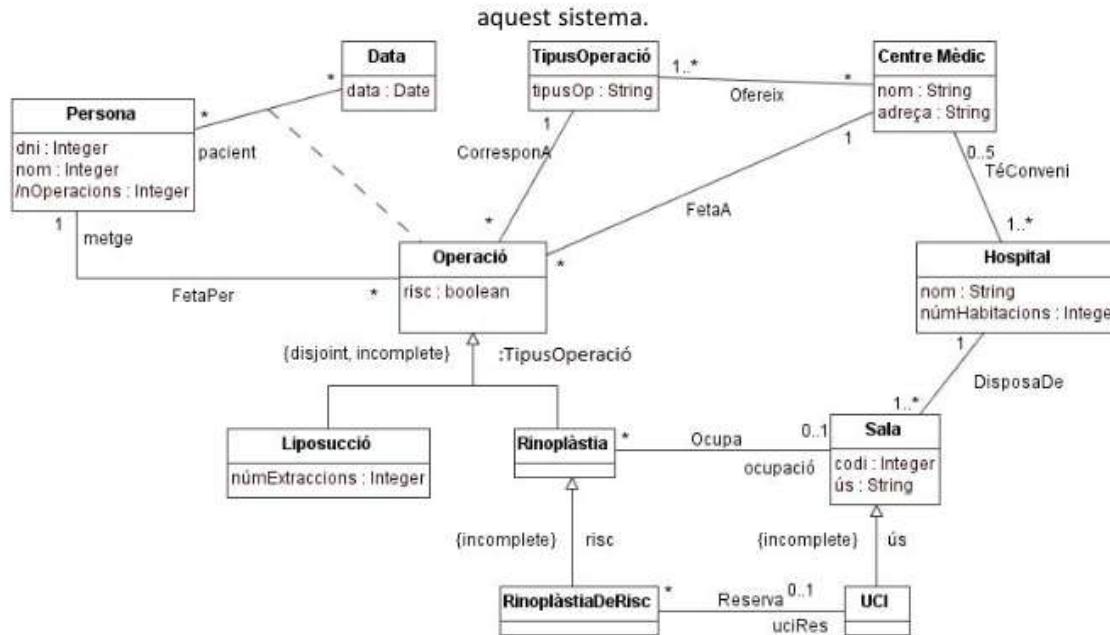


Post : crea PP {Pareja} {Concurso}



Una consorci de centres mèdics necessita un sistema software que gestioni la informació de les operacions de cirurgia estètica que realitza. A continuació disposeu de l'especificació feta per

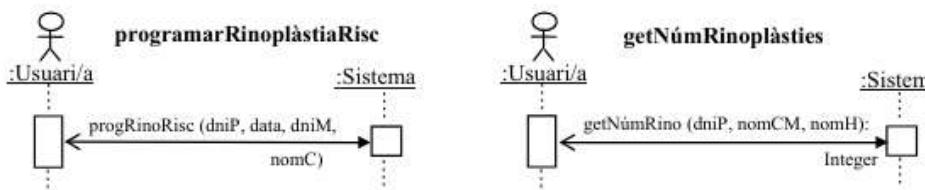
Esquema conceptual d'especificació:



Restriccions textuais

1. Claus externes: (Persona, dni), (Data, data), (TipusOperació, tipusOp), (Centre Mèdic, Nom), (Hospital, nom)
2. El metge i el pacient d'una operació no poden coincidir.
3. El tipus que correspon a una operació és un dels que ofereix el centre mèdic on es fa l'operació.
4. Un metge fa un màxim de tres operacions diàries per a un mateix centre mèdic.
5. Si una Rinoplastia té risc=cert, aleshores ha de ser del tipus 'RinoplastiaDeRisc'
6. Un hospital no pot disposar de dues sales amb el mateix codi.
7. La sala que ocupa una rinoplastia ha de ser d'un hospital que té conveni amb el centre on es fa l'operació.
8. L'UCI reservada per una rinoplastia de risc ha de ser del mateix hospital de la sala que ocupa.

Diagrama de seqüència d'esdeveniments del sistema:



Contracte de l'operació programarRinoplastiaRisc:

Operació: progRinRisc (dniP: Integer, data: Date, dniM: Integer, nomC: String)

Pre:

- La persona identificada per *dniP* existeix.
- La persona identificada per *dniM* existeix.
- El centre mèdic *nomC* existeix.

Post: Es dóna d'alta una instància de *RinoplastiaDeRisc* amb els atributs i les associacions corresponents entre la Persona *dniP* i la Data *data*.

Contracte de l'operació getNumbRinoplasties:

Operació: getNumbRino (dniP: Integer, nomCM: String nomH: String): Integer

Pre:

- La persona *dniP* existeix.
- El centre mèdic *nomCM* existeix.
- L'hospital *nomH* existeix.

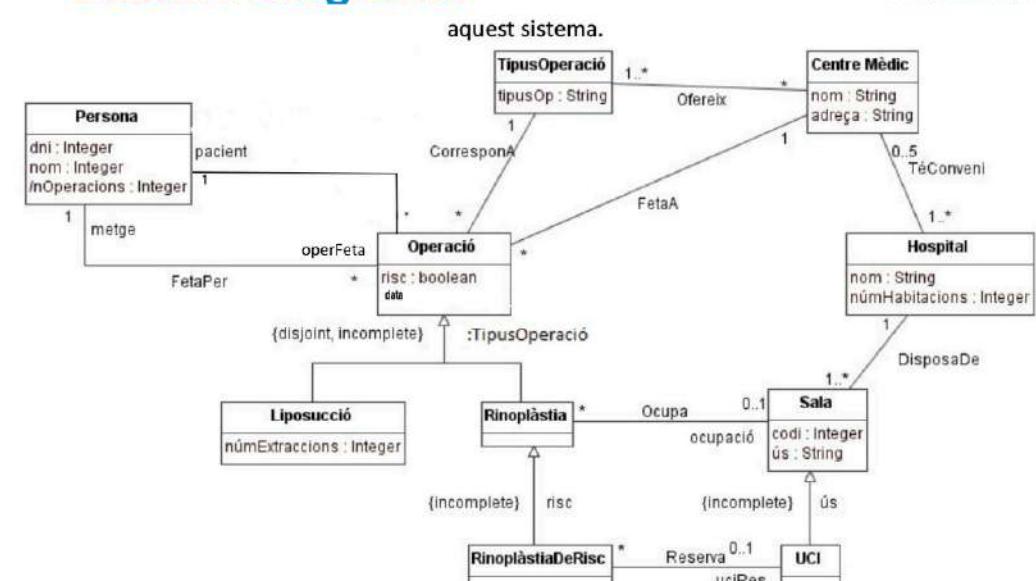
Body:

- Es retorna el nombre de vegades que la persona *nomP* ha sigut pacient d'una rinoplastia de risc feta al centre mèdic *nomCM* i que ha reservat una UCI de l'hospital *nomH* i on el metge que fa l'operació hagi sigut pacient com a mínim de 10 operacions.

Es demana:

- a) Diagrama de classes de disseny obtingut a partir de l'esquema conceptual de les dades, indicant explícitament les restriccions d'integritat que apareixen o desapareixen, i els contracte de les operacions obtinguts com a conseqüència de la traducció de l'esquema d'especificació al de disseny.

Solució Diagrama



RT: No hi ha 2 operacions amb el matrix pacient i data

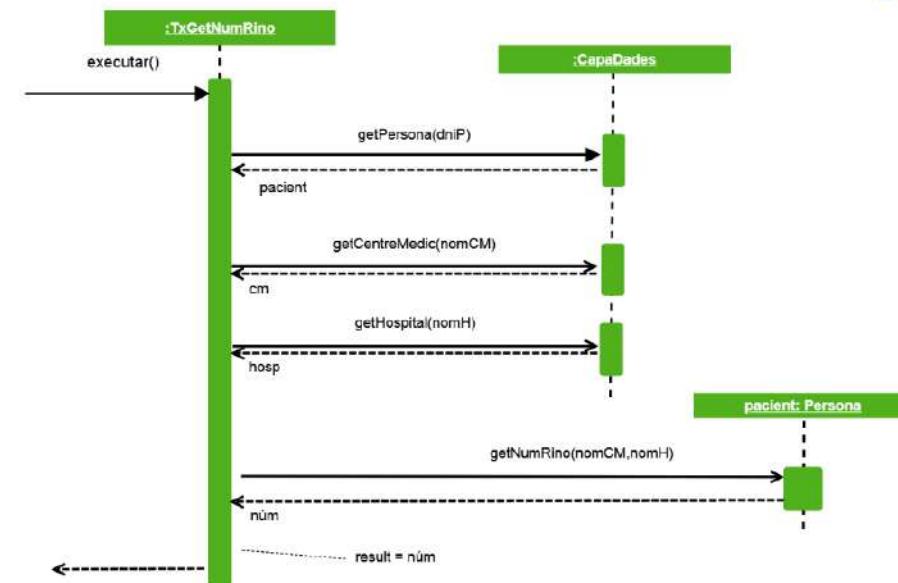
- b) Assumint que es vol utilitzar el controlador transacció, dissenyeu el diagrama de seqüència de l'operació *getNumRinoplasties* i de totes les operacions que siguin invocades en aquest diagrama de seqüència. Poseu comentaris de tot el que no hi aparegui de forma explícita. Indiqueu clarament les operacions abstractes.
- c) Assumint que es vol utilitzar el controlador transacció i que l'atribut *nOperacions* de *Persona* és materialitzat, dissenyeu el diagrama de seqüència de l'operació *programarRinoplastiaRisc* i de totes les operacions que siguin invocades en aquest diagrama de seqüència. Poseu comentaris de tot el que no hi aparegui de forma explícita. Indiqueu clarament les operacions abstractes.
- d) Indiqueu al diagrama de classes de l'apartat a) la navegabilitat resultant del vostre disseny.

Operació 1: *getNumRinoplasties*

- Operació:** *getNumRino(dniP: Integer, nomCM: String, nomH: String): Integer*
- Exc:**
 - [NoExisteixPersona]: la Persona identificat per *dniP* no existeix.
 - [NoExisteixCentreMedic]: El Centre Mèdic identificat per *nomCM* no existeix.
 - [NoExisteixHospital]: l'Hospital identificat per *nomH* no existeix.
- Body:**
 - Es retorna el nombre de vegades que la persona *nomP* ha sigut pacient d'una rinoplastia de risc feta al centre mèdic *nomCM* i que ha reservat una UCI de l'hospital *nomH* i on el metge que fa l'operació hagi sigut pacient com a mínim de 10 operacions.

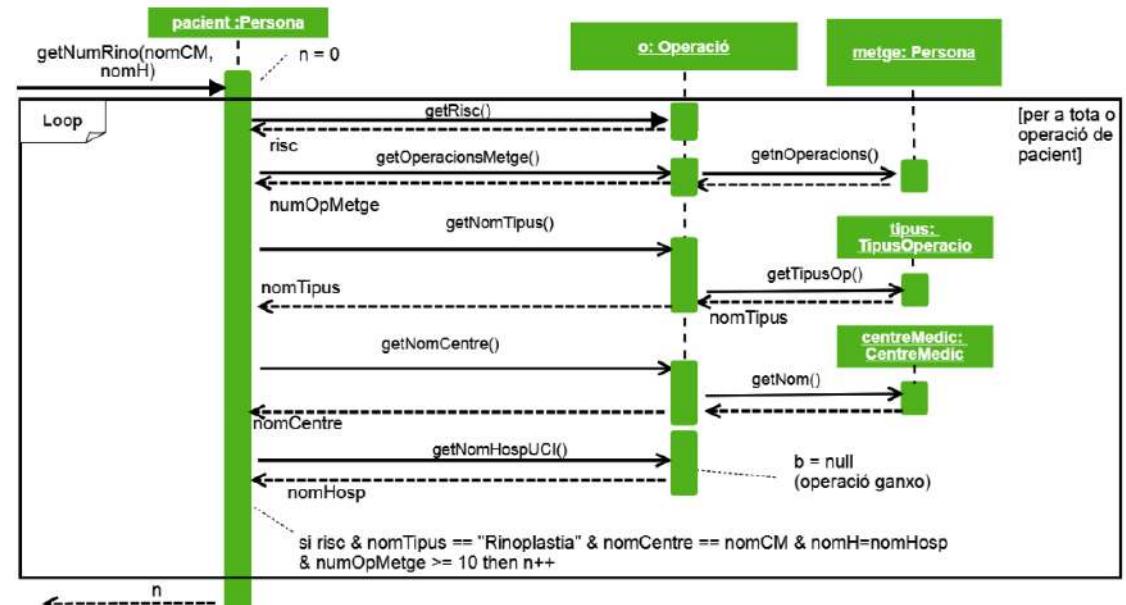
Operació 2: *ProgramarRinoplastiaRisc*

- Operació:** *progRinoRisc (dniP: Integer, data: Date, dniM: Integer, nomC: String)*
- Exc:**
 - [NoExisteixPersona]: el Pacient identificat per *dniP* no existeix.
 - [NoExisteixMetge]: El Metge identificat per *dniM* no existeix.
 - [NoExisteixCentreMedic]: El Centre Mèdic identificat per *nomC* no existeix.
 - [NoExisteixTipusRinoplastia]: La beguda identificada per *nomBeguda* no existeix.
 - [OperacióExisteix]: ja existeix l'operació amb Data, data i pacient amb *dniP*
 - [Autooperació]: el metge i el pacient son la mateixa persona (RT2)
 - [CentreMedicIncorrecte]: El centreMèdic no ofereix Rinoplastia (RT3)
 - [MetgeLimitOperacions]: El metge ja ha fet 3 operacions (RT 4)
- Post:**
 - Es dóna d'alta una instància de *RinoplastiaDeRisc* amb els atributs i les associacions corresponents entre la Persona *dniP* i la Data *data*.
 - (extra) *nOperacions* ha d'incrementar-se



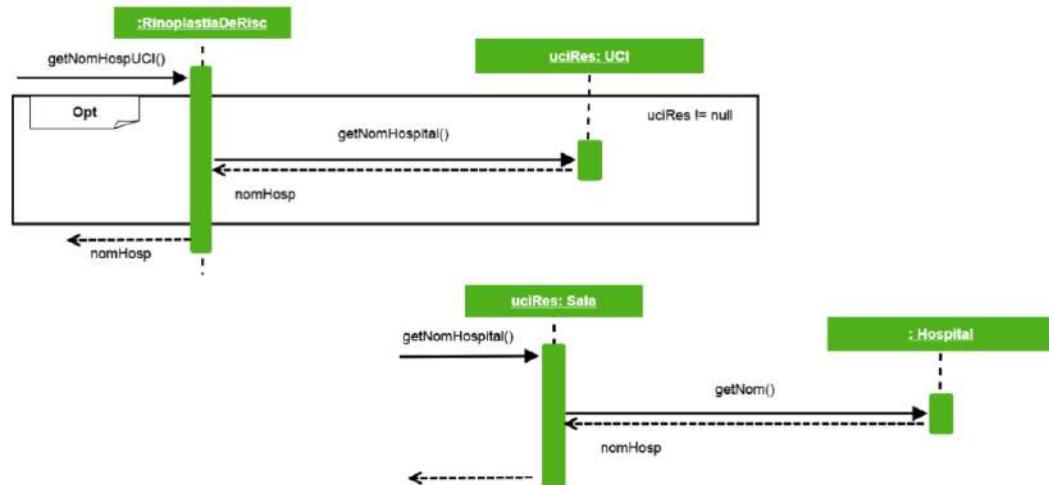
- Obtenim Persona, Centre Medic de la Capa de Dades.
 - Això pot llençar la Excepció [NoExisteixPersona], [NoExisteixCM], [NoExisteixHospital]

Persona::getNumRinoplasties

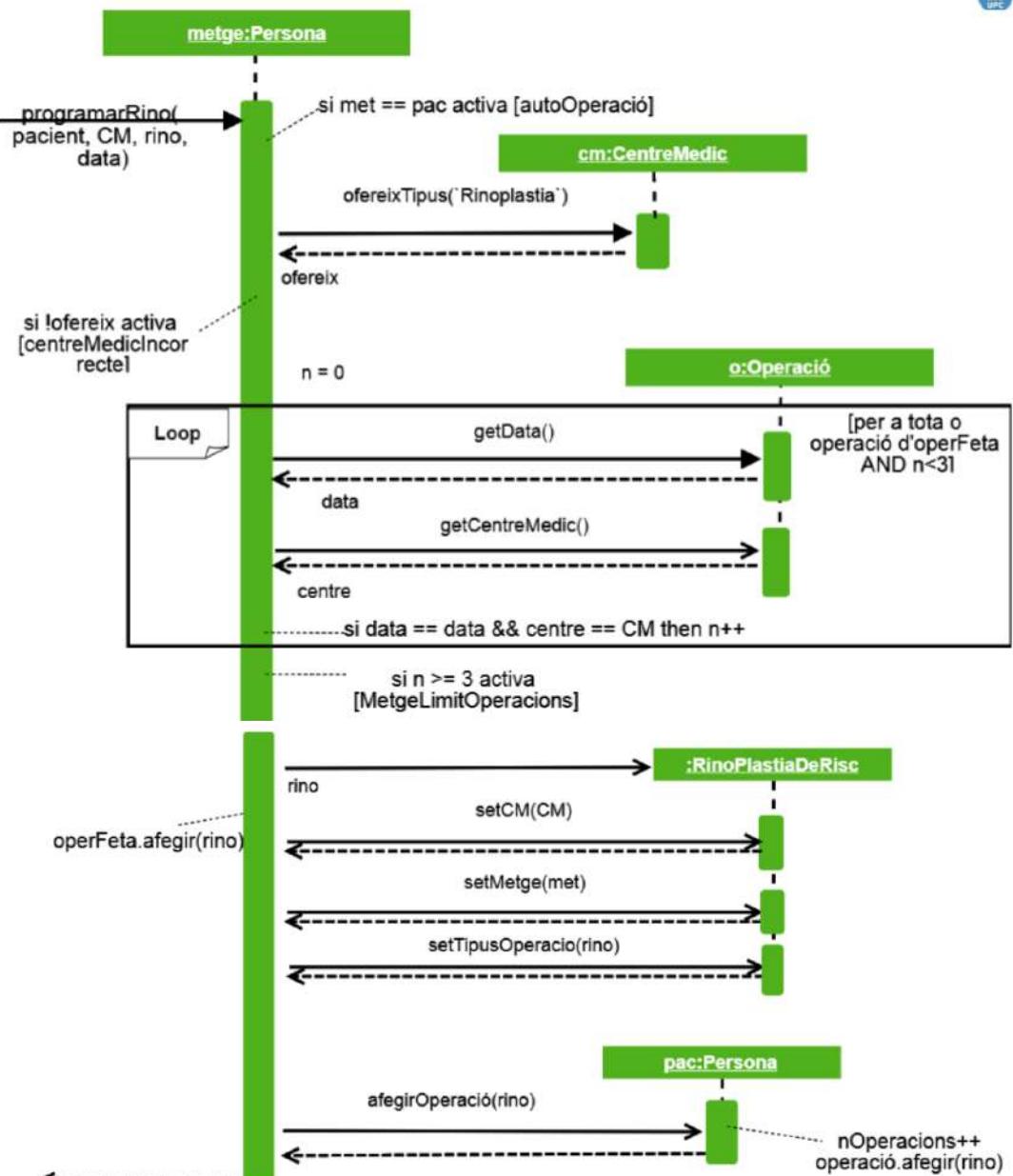
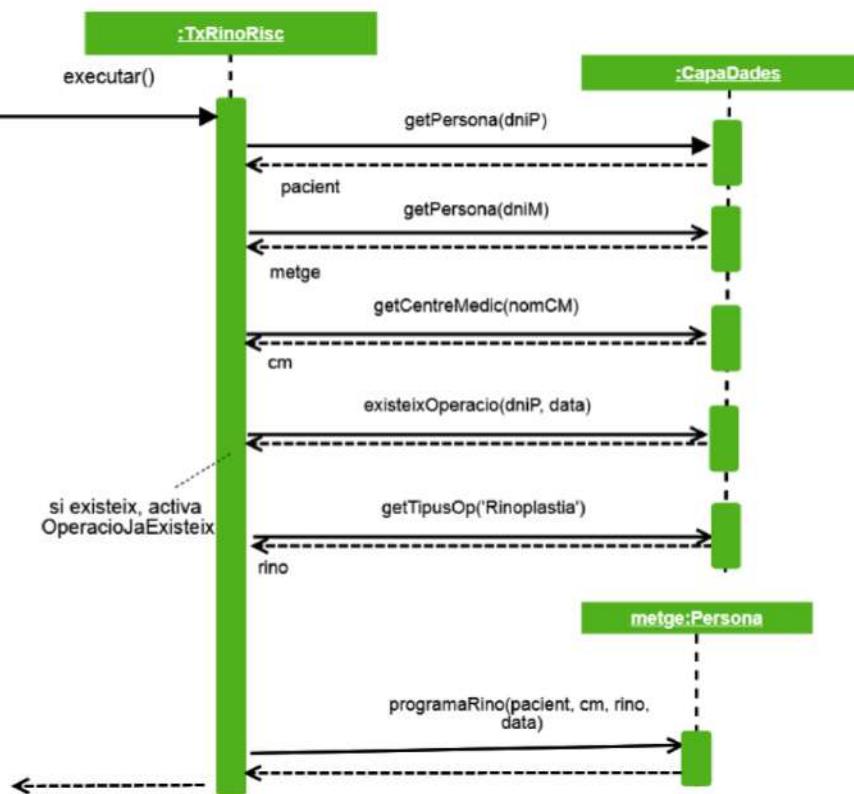


Rinoplastia de Risc::getNomHospUCI

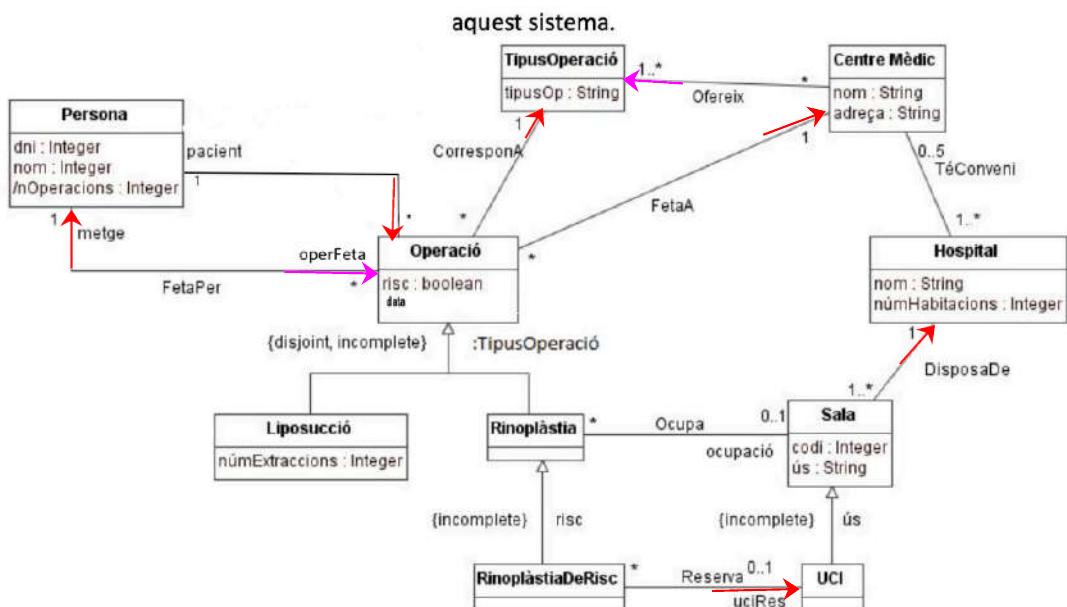
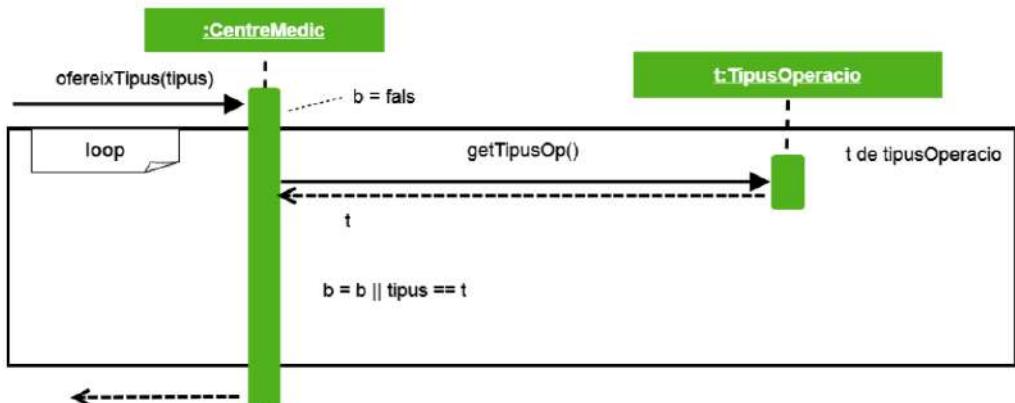
Persona::ProgramarRino (Part 1)



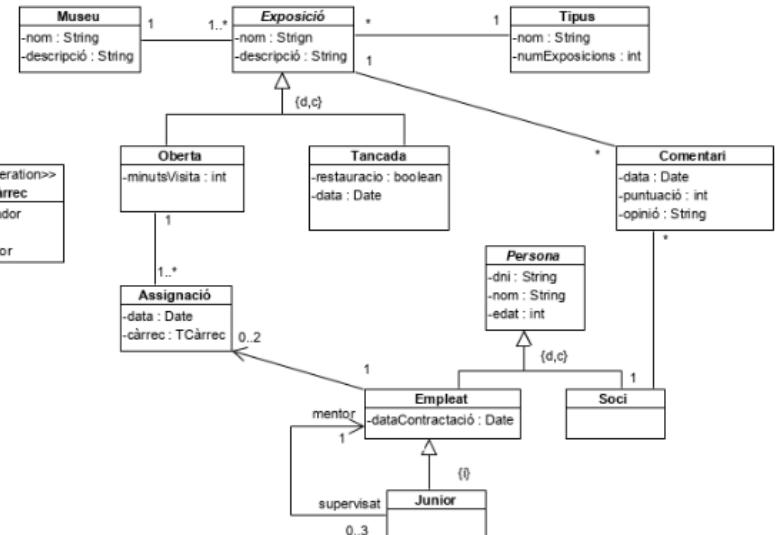
Executar



CentreMedic::ofereixTipus



Una organització que gestiona un conjunt de museus ens ha demanat que dissenyem un sistema per guardar part de la informació que manipula. El sistema emmagatzema informació de les exposicions que es fan a cada museu, en quin estat es troben (obertes o tancades) i el seu tipus. A més, també guarda informació dels empleats i socis de l'organització. En el cas dels empleats, es guarda només l'assignació actual d'aquests (és a dir, no hi ha històric), mentre que en el cas dels socis s'enregistren els comentaris que fan de les exposicions. Un mateix soci pot fer diversos comentaris d'una mateixa exposició, sempre en dates diferents.



Resticcions d'integritat textuais:

- Claus externes: (`Museu, nom`), (`Tipus, nom`), (`Exposició, nom`), (`Persona, dni`)
- 1. $0 \leq \text{puntuació} \leq 10$
- 2. Màxim hi ha un empleat amb càrrec de coordinador assignat a una exposició oberta.
- 3. La data de qualsevol assignació d'un empleat ha de ser posterior o igual a la seva data de contractació.
- 4. La data de contractació d'un junior ha de ser posterior o igual a la data de contractació del seu mentor.
- 5. No poden haver-hi dos comentaris relacionats amb la mateixa exposició i soci, i fets en la mateixa data.
- 6. No poden haver-hi dues assignacions relacionades amb el mateix empleat i exposició oberta.
- 7. Un empleat que actua com a mentor d'un altre no pot ser junior.
- 8. No poden haver-hi empleats menors d'edat (edat < 18)

Contracte de l'operació `obtenirComentarisExposicions`:

Context: `obtenirComentarisExposicions(nomM: String): Set(TupleType(nomExp:String, nomTip: String, comentaris: Set(String)))`

Excepcions: [noExisteixMuseu] El museu `nomM` no existeix.

- Body:** Per cada exposició que forma part del museu `nomM`, es retorna la següent informació:
- Si l'exposició es troba oberta, es retorna el nom de l'exposició, el nom del seu tipus, i una llista amb les opinions dels comentaris de l'exposició fets en data posterior o igual a l'assignació d'empleat més antiga que té l'exposició.
 - Si l'exposició es troba tancada, es retorna el nom de l'exposició, el nom del seu tipus, i una llista amb les opinions dels comentaris de l'exposició fets en data posterior o igual a la data des de la que està tancada.

Contracte de l'operació altaJunior:

Context: altaJunior(dniJ: String, nomJ: String, edatJ: int, dniM: String, nomE: String, data: Date, càrrec):

TCàrrec)

Excepcions:

[noExisteixExpOberta] l'exposició nomE no es troba oberta

[noExisteixEmpleat] l'empleat dniM no existeix

[jaExisteixPersona] la persona dniJ ja existeix

[menorEdat] edatJ inferior a 18

[ésJunior] l'empleat dniM és junior

[massaJuniors] l'empleat dniM ja fa de mentor a tres juniors.

[jaHiHaCoordinador] càrrecJ és "coordinador" i ja hi ha un empleat desenvolupant el càrrec de coordinador a l'exposició oberta nomE.

[dataIncorrecta] data és anterior a la data de contractació de la persona dniM

Post:

[altaJunior] Es crea una instància de Junior amb els atributs corresponents (dniJ, nomJ, edatJ, data).

[altaAssignació] Es crea una nova instància d'Assignació definida per l'empleat que s'acaba de crear dniJ i l'exposició oberta nomE, amb el càrrec corresponent càrrecJ i data data.

[assignarMentor] S'assigna l'empleat dniM com a mentor de l'empleat junior creat dniJ.

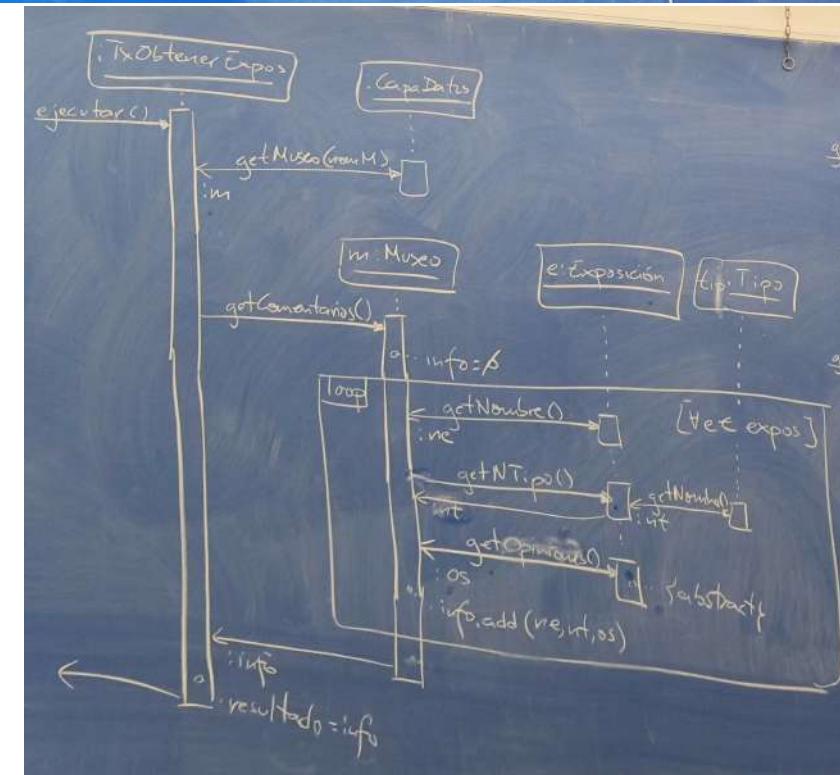
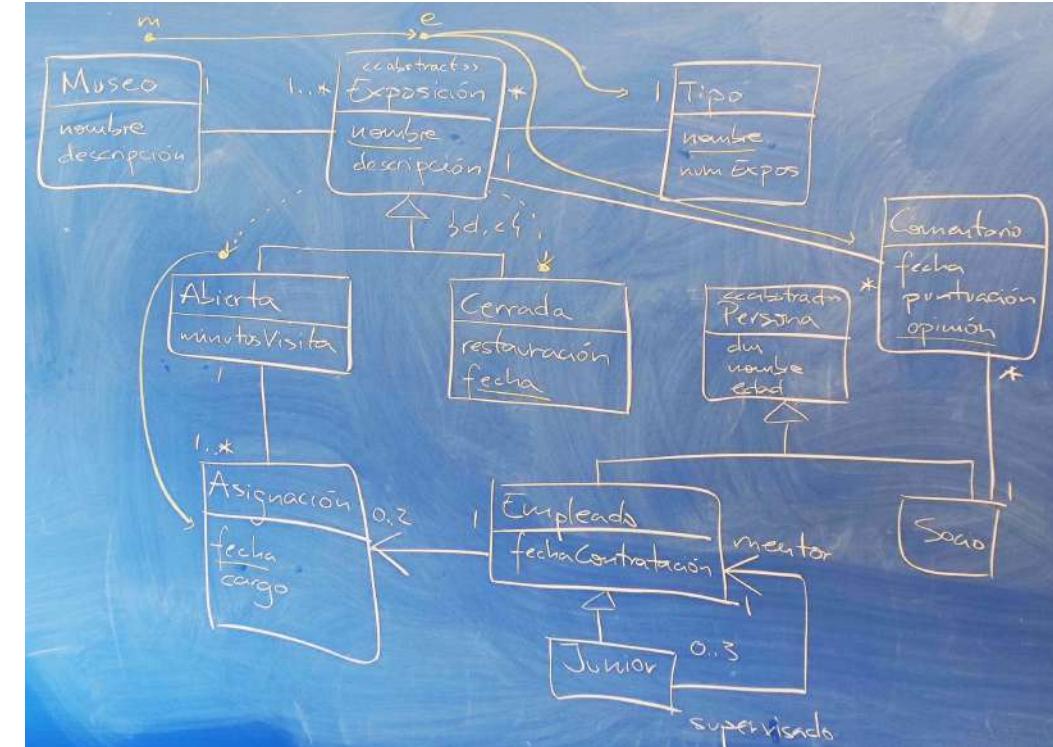
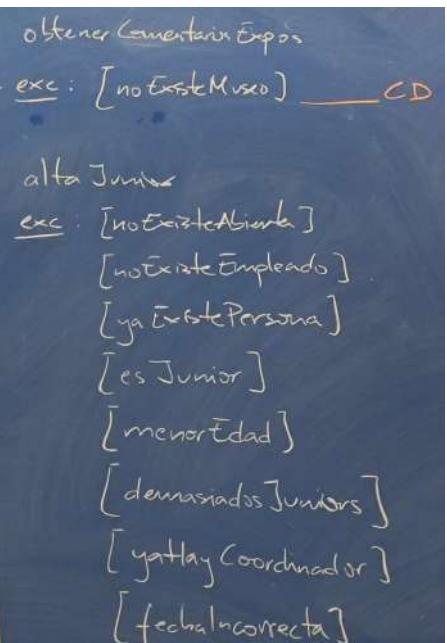
Suposant que:

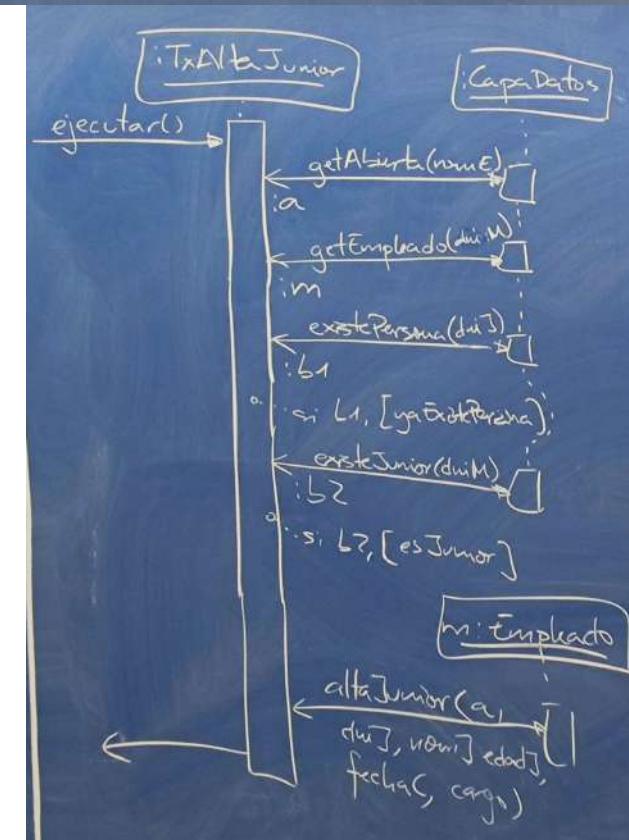
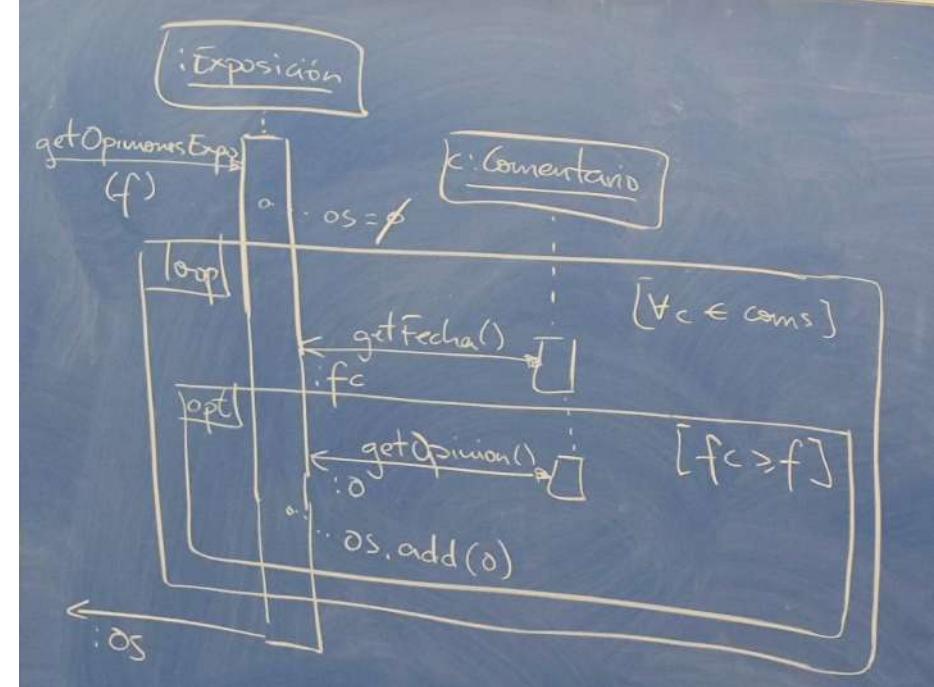
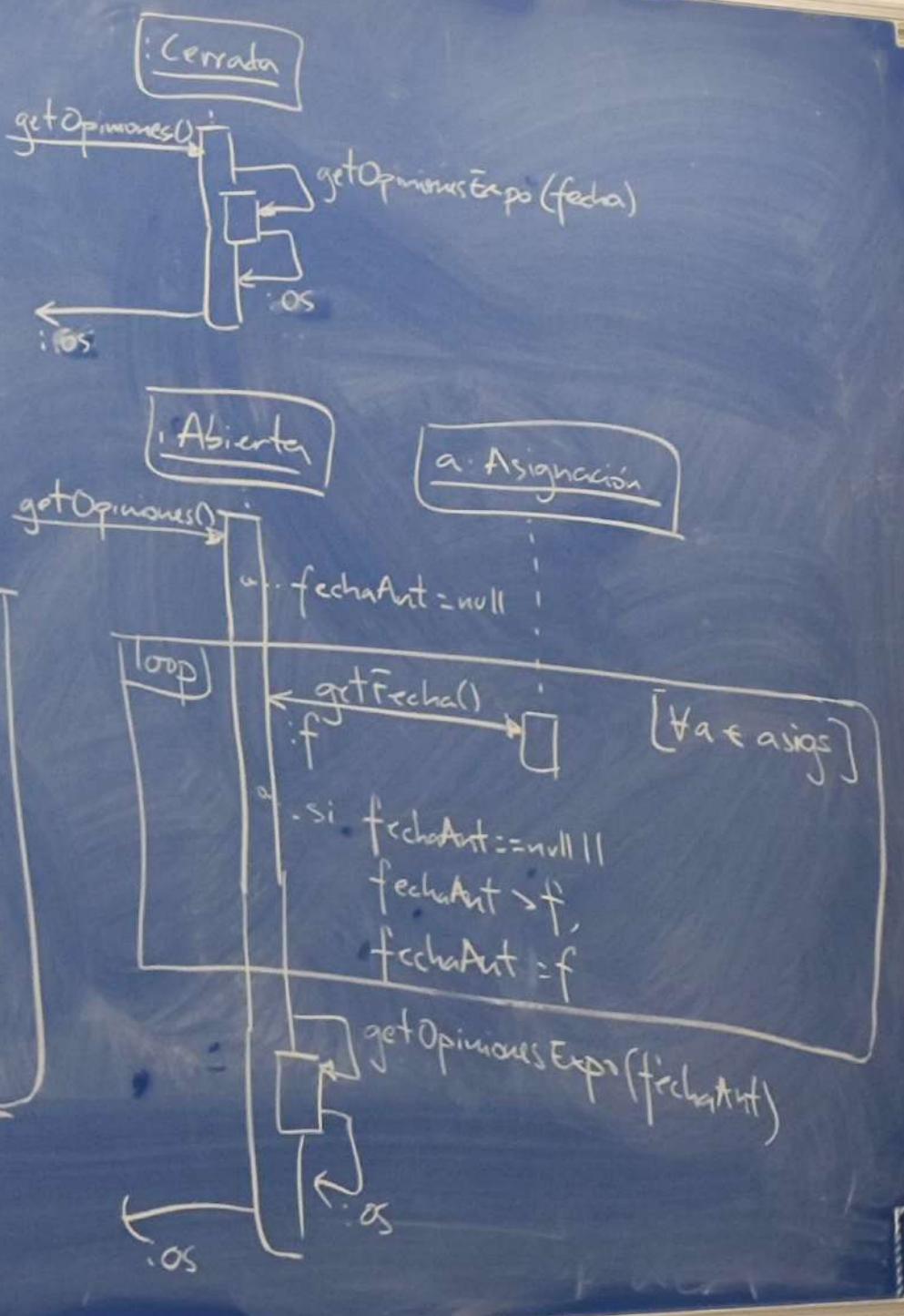
- Teniu les navegabilitats especificades al diagrama de classes de disseny inicial (i a les quals se'n poden afegir).

Es demana:

a) Diagrama de seqüència de l'operació obtenirComentarisExposicions i de totes les operacions que siguin invocades en els diagrames de seqüència. Poseu comentaris de tot el que no hi aparegui de forma explícita. Utilitzeu el controlador transacció. Indiqueu quines operacions són abstractes i en quines classes estan definides.

b) Diagrama de seqüència de l'operació altaJunior i de totes les operacions que siguin invocades en els diagrames de seqüència. Poseu comentaris de tot el que no hi aparegui de forma explícita. Utilitzeu el controlador transacció. Indiqueu quines operacions són abstractes i en quines classes estan definides. Per indicar les navegabilitats, podeu indicar-les en el mateix diagrama de classes o bé de forma textual, per exemple: Junior -> Empleat, Empleat -> Assignació.





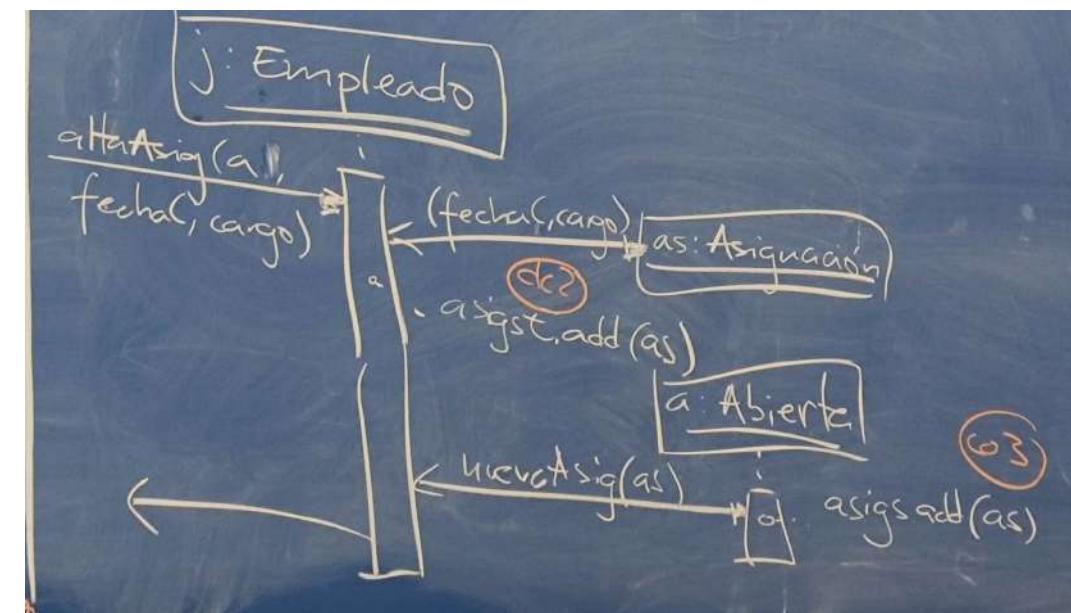
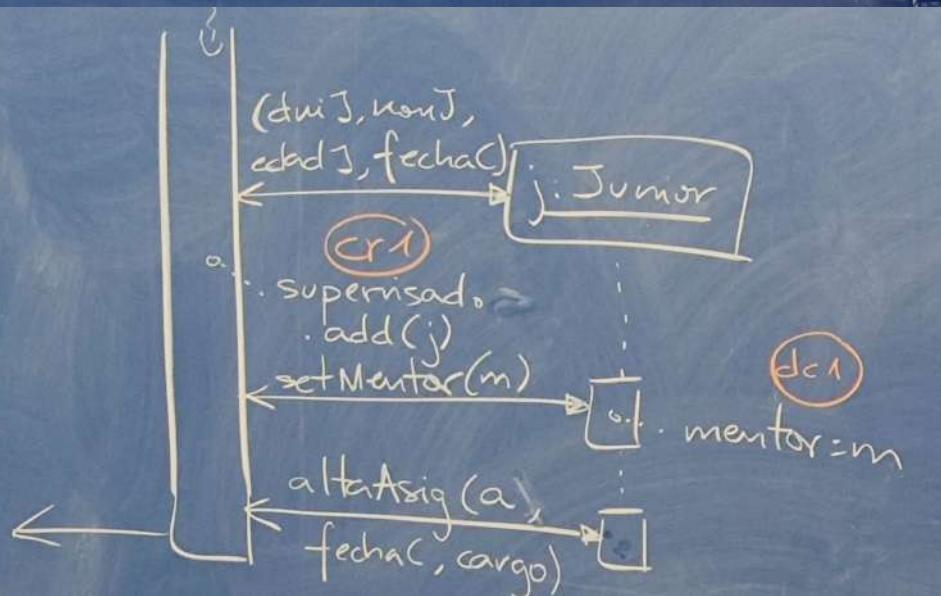
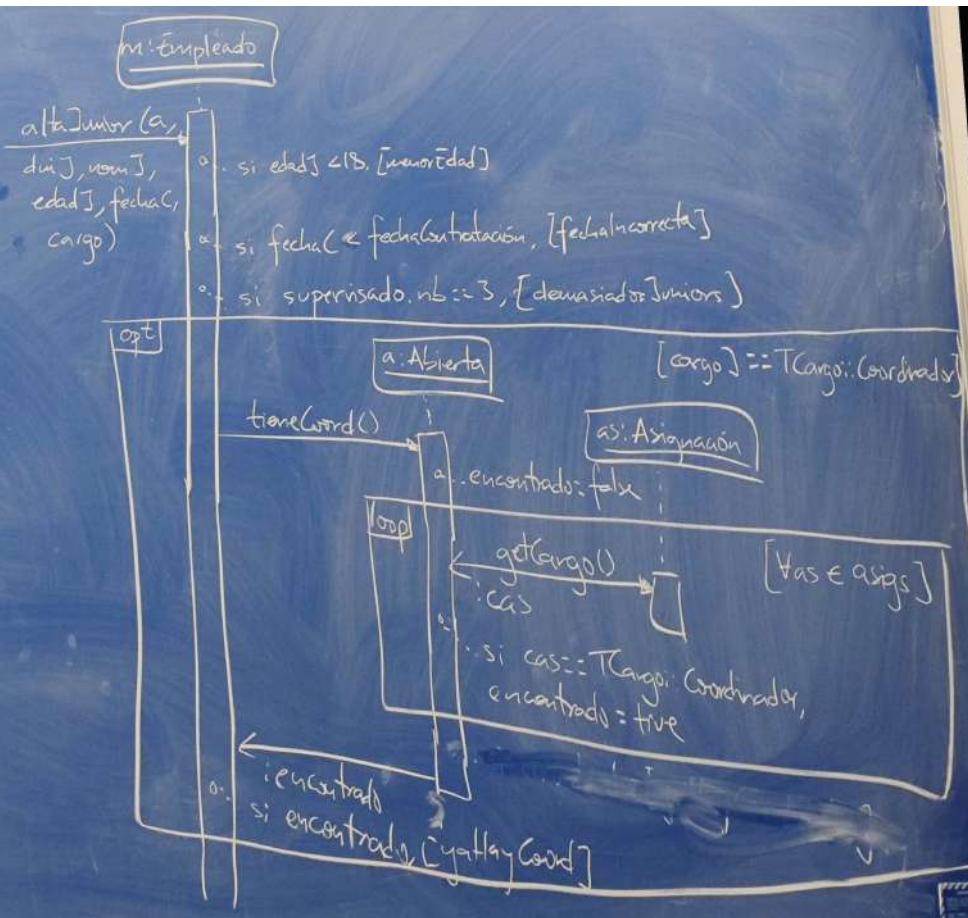
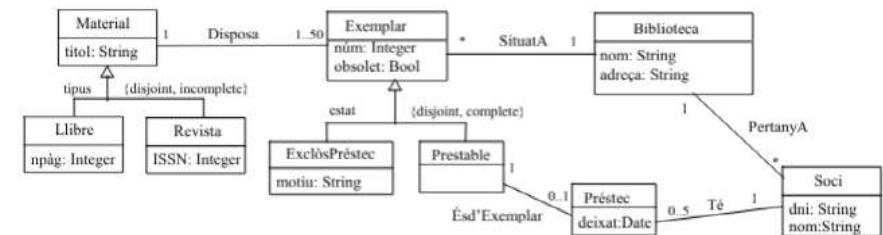


Diagrama de classes de disseny



Restriccions textuais

1. Claus externes: (Material, titol), (Biblioteca, nom), (Soci, dni)
 2. No poden existir dues revistes amb el mateix ISSN.
 3. Un material no pot tenir dos exemplars amb el mateix número.
 4. Un soci només pot tenir en préstec exemplars situats a la biblioteca a la qual pertany.
 5. Un soci no pot tenir dos préstecs diferents d'exemplars d'un mateix material.

Operació: FerPréstec (títol: String, númEx: Integer, dni: String, data: Date)

Excepcions

- [Soci no existeix] El soci amb *dni* no existeix
 - [PrestableNoExisteix] L'exemplar *númEx* del material *títol* no és prestable
 - [ExemplarObsolet] L'exemplar *númEx* del material *títol* està obsolet
 - [PréstecExisteix] L'exemplar *númEx* del material *títol* està prestat a algun soci
 - [MàximExhaurit] El soci *dni* ja té 5 préstecs
 - [SociJaTéMaterial] El soci *dni* ja té en préstec un altre exemplar del material *títol*
 - [ExemplarNoABiblio] L'exemplar *númEx* del material *títol* no està situat a la biblioteca a la qual pertany el soci *dni*

Post

- Es crea un nou Préstec per al soci *dni* amb l'exemplar *númEx* del material *títol*, que s'ha deixat a la data *data*.

Operació: *ObtenirLectorsAfins(dni: String): Set*

Excepcions:

[SociNoExisteix]: El soci amb *dni* no existeix

[SociNoTéCapPréstec]: El soci amb *dni* no té cap llibre en préstec

Body:

Per cada llibre de més de 250 pàgines que el soci *dni* té en préstec, s'obté una llista amb els dnis i els noms dels socis que tenen préstecs d'algún exemplar d'aquell llibre de la biblioteca a la que pertany el soci *dni*.

Es demana: (entregueu els apartats a i b en fulls separats)

- a) **(4 punts)** Diagrama de seqüència de l'operació *ObtenirLectorsAfans* i de totes les operacions que s'igualen invocades en els diagrames de seqüència.

b) **(4 punts)** Diagrama de seqüència de l'operació *FerPréstec* i de totes les operacions que s'igualen invocades en els diagrames de seqüència. Cal que indiqueu les navegabilitats resultants de la consultora i la creadora. Podeu entregar el full d'enunciat amb les navegabilitats marcades en el diagrama de classe si us és més còmode. **No us oblideu de posar-hi el nom.**

En ambdós casos, poseu comentaris de tot el que no hi aparegui de forma explícita. Utilitzeu el controlador de transacció. Indiqueu quines operacions són abstractes i a quines classes estan definides.

