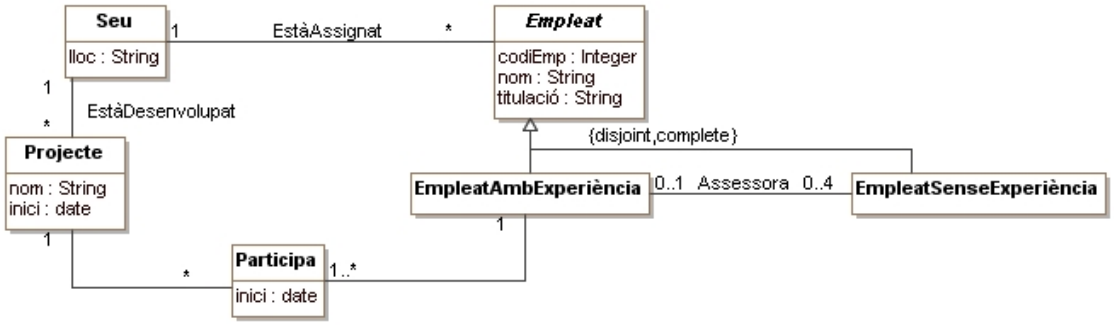
Unit 3.2. Domain Layer Design Exercises

1. Una empresa informàtica amb diferents seus a Catalunya ens ha demanat que li dissenyem una part d'un sistema software per a gestionar la participació dels seus empleats en els projectes que desenvolupa. De les seus de l'empresa, sabem la seva localització. Les seus de l'empresa desenvolupen projectes que són identificats pel seu nom. A més, també coneixem el nombre d'empleats amb experiència que participen en aquests projectes i la data d'inici del projecte. Dels empleats d'aquesta empresa sabem el codi d'empleat, el seu nom, la seva titulació i la seu a la que estan assignats. Els empleats poden ser de dos tipus: amb experiència i sense experiència. Els empleats sense experiència poden tenir l'assessorament d'un empleat amb experiència (que pot ser de la seva mateixa seu o d'una altra diferent). Els empleats amb experiència són els que participen en projectes, amb una data d'inici de participació en el projecte (la data-final de la participació no és rellevant per aquest disseny). Per simplificar podeu suposar que els empleats no canvien de tipus. Suposeu que ja s'ha fet una assignació de responsabilitats a capes i que com a resultat, es disposa d'un model de domini (perquè s'aplica Domain Model; a més, ja han aparegut algunes navegabilitats) i uns contractes d'operacions de capa de domini que es mostren tot seguit (considereu que les dues operacions són de casos d'ús diferents, amb el mateix nom de l'operació):



R.I. Textuals:
- Claus: (Seu, lloc); (Empleat, codi_emp); (Projecte, nom)
- No pot haver-hi dues participacions del mateix AmbExp en el mateix projecte.
- La data d'inici de la participació d'un empleat amb experiència en un projecte ha de ser posterior a la data d'inici del projecte.
- Els projectes en els que participa un empleat amb experiència han d'estar desenvolupats a la mateixa seu on l'empleat està assignat.

context CapaDeDomini::llistaAss (nomPr: String): Set(String)
exc *projecteNoExisteix*: El projecte identificat per *nomPr* no existeix
post result = retorna els noms dels empleats sense experiència que estan assignats a la seu on s'està desenvolupant el projecte amb *nomPr* i que són assessorats per empleats amb experiència que estan participant en el projecte *nomPr*

context CapaDeDomini::baixaEmp (codiEmp: Integer)
exc *empleatNoExisteix*: L'empleat identificat per *codiEmp* no existeix
post *eliminaAssignació:* elimina la instància de l'associació *EstàAssignat* entre l'empleat i la seu
post *eliminaParticipacióAssessorament*: si l'empleat és amb experiència, s'eliminen les seves associacions de participació, i les seves associacions d'assessorament amb

empleats sense experiència. Si l'empleat és sense experiència s'elimina l'associació d'assessorament amb l'empleat amb experiència, si és el cas
post *eliminaEmpleat*: s'elimina la instància d'empleat

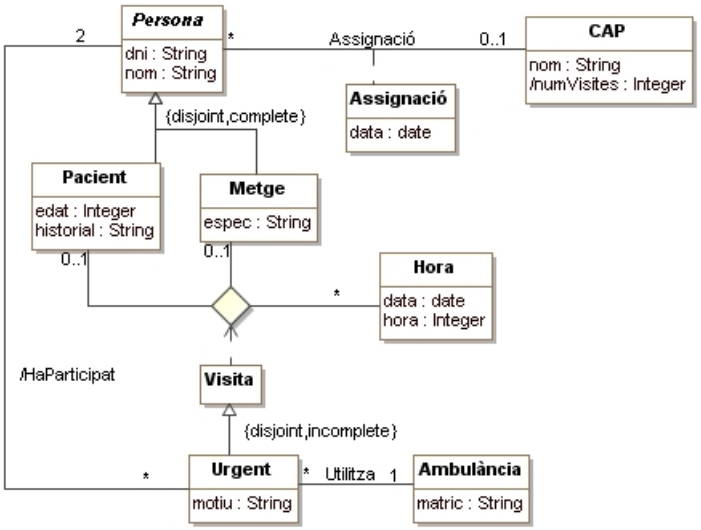
Es demana:

(a) Feu els diagrames de seqüència de les operacions anteriors. Cal indicar explícitament les comprovacions o aspectes que considereu rellevants i que no apareguin al diagrama de seqüència. Suposeu una navegabilitat inicial doble de l'associació *EstàAssignat*.

(b) Feu el diagrama de classes de la capa de domini.

2. La seguretat social ens ha demanat que li dissenyem una part d'un sistema software per a gestionar les visites que fan els pacients als centres d'assistència primària (CAP). Totes les persones que enregistra aquest sistema són metges o pacients. Aquestes persones estan assignades com a molt a un CAP (els pacients per visitar-se i els metges per visitar). D'aquesta assignació disposem de la data d'assignació. Un pacient programa una visita amb el seu metge en una certa data i hora. Les visites tenen una durada de 1 hora i poden ser de diversos tipus. En el cas de tractar-se d'una visita urgent el sistema enregistra el motiu de la visita i l'ambulància que utilitzarà per desplaçar-se al CAP. Una visita, un cop feta, no pot variar mai de tipus. A continuació disposeu de l'especificació feta per aquest sistema:

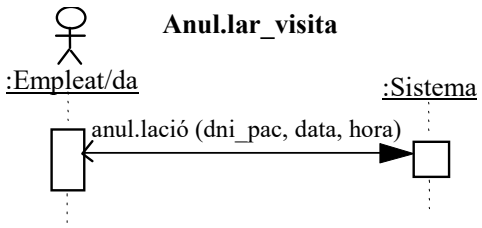
Esquema conceptual d'especificació:



R.I. Textuals:
- Claus classes no associatives: (Persona, dni); (CAP, nom); (Hora, data+hora); (Ambulància, matric)
- La data de la visita ha de ser posterior a la data d'assignació del metge i del pacient al CAP.
- El pacient i el metge que participen en una visita han d'estar assignats al mateix CAP.
- Tot CAP ha de tenir sempre més de 15 visites.

Info. derivada:
- HaParticipat: una Persona participa en un conjunt de visites urgents.
- numVisites: és el nombre de visites totals del CAP.

Diagrama de seqüència d'esdeveniments del sistema:



Contracte de l'operació anul.lació:

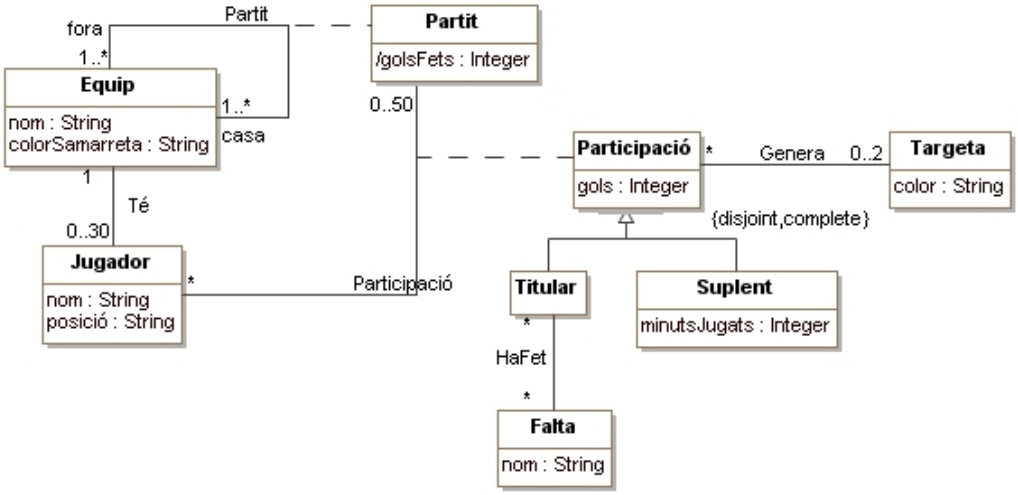
context anul.lació (dniPac: String, data: Date, hora: Integer)
pre *visitaExisteix:* la visita existeix
pre *visitaNoUrgentONoProblemaCor:* la visita no és urgent o, si ho és, no té com a motiu "Problema de Cor"
post *eliminaVisita:* dóna de baixa l'ocurrència de l'associació *Es_visita*
post *eliminaUtiliza:* si la visita és urgent es dóna de baixa l'associació *Utilitza*

Es demana:

(a) Feu el contracte de la capa de domini considerant un disseny extern sense cap tipus de desplegable (és a dir, per a cada operació, la informació s'entra directament en camps de text o numèrics, tota de cop), patró Domain Model a la capa de domini i assignació de totes les responsabilitats que apareixen en els contractes anteriors a la capa de domini, tret de les responsabilitats de clau que s'assignen a la capa de gestió de dades. Considereu que la informació derivada es materialitza i que les navegabilitats de *Utilitza* i *HaParticipat* és doble.
(b) Feu els diagrames de seqüència de les operacions resultants de la capa de domini. Cal indicar explícitament les comprovacions o aspectes que considereu rellevants i que no apareguin al diagrama de seqüència.
(c) Feu el diagrama de classes de la capa de domini.
(d) Suposeu ara que decidim que l'associació *HaParticipat* es calcula. Feu els diagrames de seqüència de les operacions resultants del càlcul.

3. La Federació Internacional de Futbol ens ha demanat que dissenyem dos casos d'ús per gestionar la informació de les seves competicions. Aquest federació disposa de la informació dels partits que es juguen i dels jugadors de que disposa cada equip. De cada equip es coneix el nom, i el color de la samarreta titular amb la que juguen. Dels jugadors es disposa del seu nom i de la seva posició en el camp, i dels partits es coneix els gols totals fets pels dos equips. Per cada jugador que participa en un partit també es coneix els gols que ha fet en aquell partit, les targetes que li han mostrat i si ha jugat com a titular o suplent. Per les participacions com a suplent es coneix el nombre de minuts jugats i per les participacions com a titular les faltes fetes (es pot considerar que els jugadors suplents no fan faltes). Els casos d'ús que volem dissenyar permeten traspassar a un jugador a un altre equip i obtenir els jugadors que fan una determinada falta quan juguen a casa. A continuació disposeu de l'especificació feta per aquest sistema:

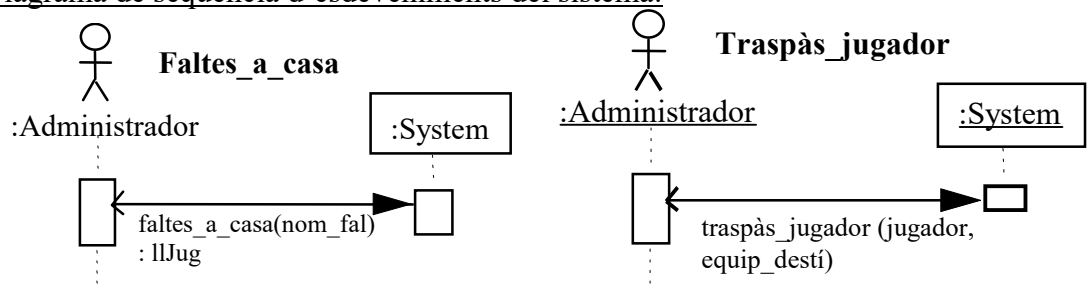
Esquema conceptual d'especificació:



R.I Textuals:
- Claus classes no associatives: (Equip, nom); (Jugador, nom); (Targeta, color); (Falta, nom_fal).
- Un equip no pot jugar amb ell mateix.
- Un jugador només pot participar en partits en els que juga el seu equip.

Info. derivada:
- golsfets: és el nombre de gols fets per tots els jugadors que han participat en un partit.

Diagrama de seqüència d'esdeveniments del sistema:



context faltes_a_casa(nom_fal: String): Set(String)
pre *existeixFalta:* existeix la falta *nom_fal*
post: result = noms de jugadors que han fet la falta *nom_fal* en algun dels partits jugats a casa

context traspàs_jugador(jug: String, equip_destí: String)
pre *existeixJugador:* existeix el jugador *jug*
pre *existeixEquipDestí:* existeix l'equip *equip_destí*
pre *jugadorNoJugaDestí:* el jugador *jug* no juga a l'equip *equip_destí*
post: 2.1 s'eliminen totes les associacions participa del jugador en el seu equip origen, tant si era suplent com titular
2.2 s'eliminen les associacions entre participació i targetes
2.3 s'eliminen les associacions entre la participació com a titular i les faltes fetes
2.4 s'elimina l'associació Té entre l'equip origen i el jugador
2.5 es crea un nova associació Té entre l'equip destí i el jugador

Es demana:

(a) Feu l'especificació de la capa de domini considerant un disseny extern sense cap tipus de desplegable (és a dir, per a cada operació, la informació s'entra directament en camps de text o numèrics, tota de cop), patró Domain Model a la

capa de domini, i assignació de totes les responsabilitats que apareixen en els contractes anteriors a la capa de domini, tret de les responsabilitats de clau que s'assignen a la capa de gestió de dades. Considereu que l'atribut derivat *golsfets* d'un partit és calculat.

(b) Feu els diagrames de seqüència de les operacions resultants de la capa de domini. Cal indicar explícitament les comprovacions o aspectes que considereu rellevants i que no apareguin al diagrama de seqüència. Suposeu que la navegabilitat de les associacions que apareixen a la capa de domini és doble.

(c) Feu el diagrama de classes de la capa de domini.

4. Es disposa d'accés a un servei de conversió de divises, GimmeTheRate.com, que ofereix dues operacions amb la descripció següent:

```
context GimmeTheRate::newRate(cFrom: String, cTo: String, excRate: Float)
pre 1.1: cFrom <> cTo
post 2.1: enregistra que la proporció entre les divises és: cFrom = cTo*excRate
```

```
context GimmeTheRate::convert(cFrom: String, cTo: String, amount: Integer): Float
exc 1.1 canvi-desconegut: no es disposa de conversió entre les divises
post 2.1: result = amount × (proporció entre les divises cFrom i cTo)
```

Una agència de viatges catalana vol donar als seus clients l'oportunitat de demanar el preu de les seves transaccions en la divisa que vulguin. Després d'explorar el mercat, l'arquitecte del software ha decidit que GimmeTheRate.com és la millor oferta disponible. L'arquitecte decideix de desenvolupar el mètode següent:

```
context CapaDomini::calculaPreu(v: Viatge, cTo: String): Float
exc 1.1 canvi-desconegut: no es disposa de conversió entre euros i cTo
post 2.1: retorna el preu del viatge v expressat en la divisa cTo
```

Aquest mètode es recolça en la classe Viatge, ja existent en el sistema, que disposa d'un mètode d'interès:

```
context Viatge::calculaPreu(): Float
post 2.1: retorna el preu del viatge en euros
```

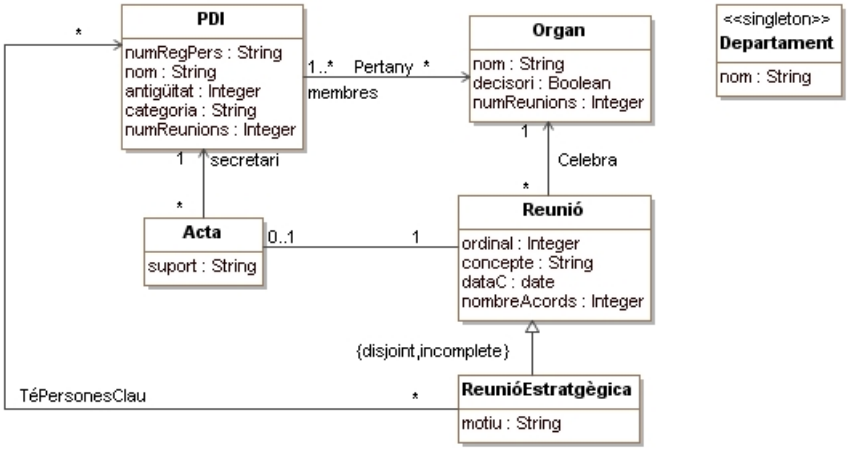
Es demana:

(a) Dissenyeu el mètode *calculaPreu* usant el patró Domain Model i controladors transacció a la capa de domini. Mostreu en un diagrama de classes totes les classes, operacions.

(b) L'agència vol ampliar el seu mercat i per això decideix vendre la seva aplicació a qualsevol altra agència del món que la vulgui comprar; cada agència treballa en la divisa pròpia del seu país. Però es troba amb un problema de manteniment, atès que l'aplicació treballa implícitament en euros. Feu les modificacions que calguin (mínimes, i que siguin robustes envers futurs canvis) per corregir aquest error de disseny.

(c) Els arquitectes del software de l'agència, després de diverses converses amb els dissenyadors de *GimmeTheRate*, verifiquen que el canvi de divisa només s'actualitza dues vegades al dia, concretament a les 00:00 i 12:00 GMT. Contràriament, una agència de viatges típica necessita calcular canvis de divisa milers de vegada al dia. Comenteu si creieu que es possible efectuar alguna millora d'eficiència de l'aplicació de l'agència i en cas afirmatiu, implementeu-la. Suposeu que existeix una operació *dataAvui(): Date* que dóna la data GMT actual del sistema (dia+hora), i una altra operació *comparaDates(d1: Date, d2: Date): Bool* que retorna cert si d1 és menor que d2.

5. Un departament de l'UPC ens ha demanat que dissenyem un sistema per gestionar les reunions dels diferents òrgans del departament. Els òrgans poden ser decisoris i enregistren en nombre de reunions fetes i les reunions registren el nombre d'acords que s'hi van prendre i l'acta quan s'ha fet. Les reunions estràtegiques enregistren el motiu i els PDI que hi van ser persones claus a la reunió. Noteu com són les navegabilitats de partida.



Claus: (PDI: numRegPers, Òrgan: nom, Reunió: Òrgan::nom+ordinal+concepte)

En fer l'especificació del sistema es va identificar el servei de directori de l'UPC. Aquest servei emmagatzema les dades personals de tots els PDIs i pot ser utilitzat per obtenir les dades dels PDIs del departament.
A l'hora de fer el disseny del sistema hem de considerar l'existència d'un servei dedicat SvDMS de gestió documental que enregistra les actes de les reunions. Aquest servei ens pot ajudar a dissenyar i implementar més ràpidament el sistema. Pels dos serveis anteriors, us mostrem el conjunt complet d'operacions, de manera que el servei només és capaç de gestionar la informació que li entra en aquestes operacions.

context SvDMS::novaReunió (nomO: String+concR: String+ordR: Int+dataR: Date+nAcR: Int)

exc reunió-existeix: la reunió nomO+concR+ordR existeix

post es dóna d'alta una reunió amb nomO , concR, ordR, dataR i nAcR

context SvDMS::novaActa (nomO: String+concR: String+ordR: Int+nrpP: String+supR: String)

exc reunió-no-existeix: la reunió nomO+concR+ordR no existeix

exc ja-té-acta: la reunió nomO+concR+ordR ja té acta

post es dóna d'alta una acta per a la reunió nomO+concR+ordR amb secretari nrpP i suport supR

context SvDMS::obtéSecretarisMés10ActesÚltimAny (): Set(TupleType(nrp:String + reus: Set(nomO:String, ordR:Integer, descR:String)

post obté els secretaris d'actes a 10 o més reunions celebrades l'últim any i per cada un, les reunions de les que han estat secretaris.

context SvDMS::obtéReunions (): Set(TupleType(nomO: String+concR: String+ordR: Int+dataR: Date+nAcR: Int))

post obté totes les reunions.

context DirectoriUPC::nouPDI (nrp: String+nomP: String+catP: String+dep:String)

exc PDI-existeix: el PDI amb nrp existeix

post es dóna d'alta un PDI amb número de registre personal nrp, nom nomP, departament dep i categoria catP

context DirectoriUPC::nouCurs ()

post incrementa en 1 l'antigüitat de tots els PDIs del sistema

context DirectoriUPC::obtéPerCategoria (cat:String): Set(TupleType(nrp:String, nom:String, dep:String, antig:Integer)

post obté les dades de tots els PDI de la categoria cat.

context DirectoriUPC::obtéPerDept (dep:String): Set(TupleType(nrp:String, nom:String, cat:String, antig:Integer)

post obté les dades de tots els PDI del departament dep.

Estem interessats en les dues operacions següents de la capa de domini del nostre sistema:

context CapaDeDomini::novaReunióEstratègica(nomO: String, concR: String, ordR: Int, motR: String, dataR: Date, nbA: Int)

exc òrgan-no-existeix: l'òrgan nomO no existeix

exc reunió-existeix: la reunió nomO+concR+ordR existeix

post es dóna d'alta una reunió estratègica per a l'òrgan nomO amb concepte concR i ordinal ordR, amb motiu motR, amb data dataR i nombre d'acords nbA
post s'incrementa el nombre de reunions enregistrat per a l'òrgan nomO

context CapaDeDomini::sobrecarregatsInútilment(): Set(nrp: String)

post retorna el conjunt de números de registre personal de PDIs que tenen la categoria Titular d'Universitat i han estat secretaris d'actes a 10 més reunions no estratègiques celebrades l'últim any

Es demana:

(a) Diagrama de classes del disseny del sistema d'informació incloent-hi els atributs. No cal declarar les operacions auxiliars del sistema. Indiqueu-ne els acoblaments i navegabilitats finals.

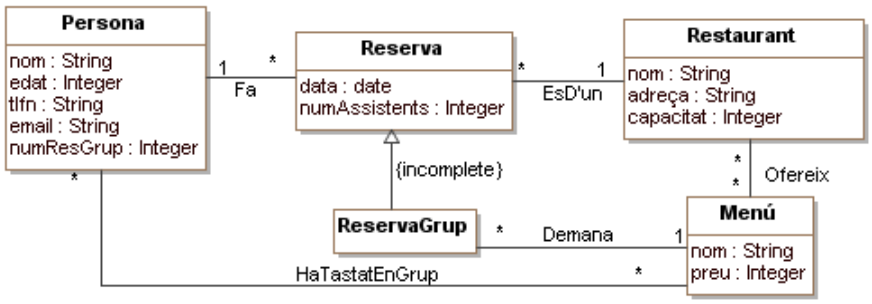
(b) Diagrama de seqüència de les dues operacions de la capa de domini del sistema, fent servir Domain Model i Data Mapper, i de totes les auxiliars que hi apareguin, tret de les operacions que quedin assignades als serveis.

(c) Com canviaria la teva solució si el servei SvDMS tingues disponible l'operació:?

context DirectoriUPC::obtéPerDepiCat (dep:String, cat:String): Set(TupleType(nrp:String, nom:String, antig:Integer)
post obté les dades de tots els PDI del departament dep i categoria cat.

Com a criteris de disseny en aquest problema, sobretot volem minimitzar el nombre d'invocacions remotes mantenint els criteris habituals (canviabilitat, portabilitat, reusabilitat). També es vol minimitzar la redundància de les dades.

6. Una cadena de restaurants que ofereix menús per sopars ens ha demanat que li dissenyem una part d'un sistema software per gestionar les seves reserves. L'esquema conceptual (de l'especificació) es mostra a continuació:



R.I. Textuals:
- Claus: (Persona, nom); (Restaurant, nom); (Menú, nom); (Reserva, Persona::nom+data);
- Els menús de les reserves de grup han de ser menús oferts pel restaurant on s'ha fet la reserva.
- La capacitat d'un restaurant ha de ser més gran que el sumatori dels assistents de les reserves previstes per aquell restaurant en una data determinada.
- El numAssistents d'una reserva de grup ha de ser més gran que 5.
- Els menús que ha tastat en grup una persona són els mateixos que ha reservat.
- El numResGrup d'una persona és igual al número de reserves de grup que ha fet la persona.
- Tots els atributs de tipus integer han de ser positius.
- Altres restriccions no rellevants per l'exercici.

La capa de domini ofereix l' operació següent:

context novaReservaGrup(nomClient:String, nomRest:String, dr:date, nAss:Integer, nomMenú:String): Integer
pre *client-existeix*, *data-ok*, *més-5assistents*, *reserva-no-existeix*
exc *restaurant-no-existeix*: el restaurant amb *nomRest* no existeix.
exc *menu-no-ofert*: el restaurant *nomRest* no ofereix el menú *nomMenú*.
exc *restaurant-sense-lloc*: el restaurant *nomRest* no té disponibilitat pel *nAss* a la data *dr*.
post *reserva-grup-creada:* es crea una reserva de grup i es formen totes les associacions amb la persona, el restaurant i el menú.
post *numResGrup-incrementat:* s'incrementa el *numResGrup* de la persona *nomClient*.
post *haTastat-creat:* si la persona no ha tastat el menú, es crea una instància de *HaTastatEnGrup* entre la persona *nomClient* i el menú *nomMenú*.
post *result*= número de places disponibles del restaurant *nomRest* a la data *dr* (capacitat del restaurant – número de assistents (de les reserves) per la data *dr* - *nAss*).

La cadena de restaurants té un sistema CRM per gestionar la informació dels seus clients. Per integrar el sistema CRM a la nostra arquitectura s'ha definit el servei *SvCRM*. Cada restaurant disposa d'un servei que conté la informació del restaurant i dels menús que ofereix (dels menús només es guarda el nom). Podeu suposar que el

servei d'un restaurant que té nom *nom* es diu *Svnom*. A més, es decideix llogar un servei genèric de gestió de reserves (aquest servei no distingeix entre les reserves de grup i les que no ho són). A continuació disposeu de les operacions proporcionades per cada servei (no es poden afegir operacions als serveis ni modificar les existents):

context SvCRM::obteDadesClient (inout dCl: DTOClient)
exc client-no-existeix: el client *dCl.nom* no existeix
post assigna l'edat, tlfn i email del client amb nom *dCl.nom* a *dCl.edat, dCl.tlfn* i *dCl.email*
**La classe DTOClient té com atributs nom, edat, tlfn i email.*

context Svnom::obteDadesRestaurant (): DTORestaurant
post retorna el nom, l'adreça, capacitat i menús que ofereix el restaurant a *dR.nom*, *dR.adreça*, *dR.capacitat* i *dR.menús*
**La classe DTORestaurant té com atributs nom, adreça, capacitat i un conjunt de noms de menús.*

context SvGestorReserves::creaReserva (dRes:DTOReserva)
exc reserva-existeix: la reserva de la persona *dRes.nom* i data *dRes.data* existeix
post es dona d'alta la reserva

context SvGestorReserves::obteDadesReserva(inout dRes:DTOReserva)
exc reserva-no-existeix: la reserva de la persona *dRes.nom* i data *dRes.data* no existeix
post retorna el nom del restaurant i el número de assistents de la reserva identificada per *dRes.nom* i *dRes.data*

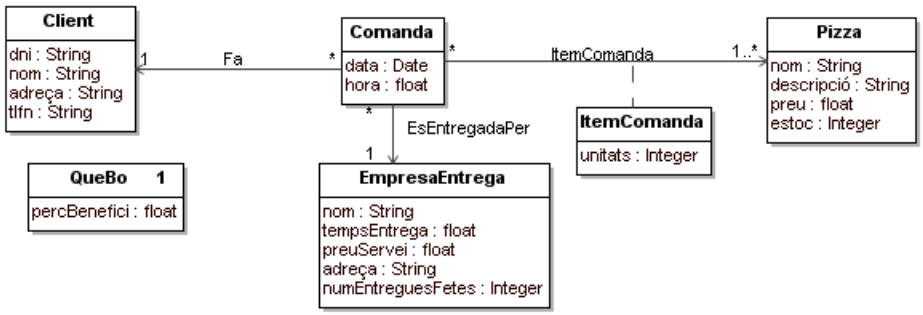
context SvGestorReserves::obtenirOcupacióxData(inout dOcup: DTOOcupacióReserva)
post *dOcup.numAssistents*= sumatori dels assistents de les reserves del restaurant *dOcup.nomRes* en data *dOcup.dt*.
**La classe DTOReserva té com atributs nom persona, nom restaurant, data i número d'assistents.*
**La classe DTOOcupacióReserva té com atributs nom restaurant, data i número d'assistents.*

Es demana:

(a) Diagrama de classes de la capa de domini del sistema de reserves, incloent-ne els atributs (però no les operacions). Suposeu Domain Model, Data Mapper, i controlador Transacció. Es vol minimitzar (per sobre d'altres criteris) la informació redundant entre els serveis i el sistema que volem dissenyar. Justifica el diagrama de classes obtingut. Suposeu la navegabilitats doble de l'associació entre *Persona* i *Reserva*, la navegabilitat simple de *HaTastatEnGrup* de *Persona* a *Menú*, la navegabilitat simple de *EsD'un* de *Reserva* a *Restaurant* i la navegabilitat simple de *Demana* de *Reservagrup* a *Menú*.

(b) Diagrama de seqüència de l'operació *novaReservaGrup*.

7. Volem dissenyar un sistema software (QueBo.com) per gestionar les comandes de la pizzeria QueBo. Aquesta pizzeria vol oferir als seus clients un sistema per demanar pizzes per internet. Els clients registrats en aquest sistema podran fer comandes de les pizzes que ofereixi la pizzeria. Les pizzes tenen un nom que les identifica, la descripció, el preu de venda al públic i l'estoc. Les comandes les entrega una empresa. De les empreses d'entrega coneixem el nom, el temps d'entrega estimat, el preu del servei d'entrega, l'adreça i el número d'entregues fetes. La pizzeria QueBo enregistra el percentatge de benefici que obté de la venda de les pizzes. A continuació disposeu del diagrama de classes del sistema:



Restriccions textuals:

RT1. Claus: (Client, dni); (Comanda, Client::dni+data+hora); (Pizza, nom); (EmpresaEntrega, nom)

RT2. La data i hora d'una comanda són correctes.

RT3. Tots els atributs integers i floats tenen valors positius.

...altres restriccions no rellevants per al problema

La pizzeria QueBo ha decidit reorganitzar el seu negoci i delegar l'elaboració de les pizzes a el proveïdor extern JoLaFaig. La pizzeria ofereix als seus clients un subconjunt de les pizzes que elabora el proveïdor. També ha arribat a un acord amb un conjunt d'empreses que gestionaran l'entrega de les pizzes a domicili. De fet, la pizzeria s'encarregarà només de la gestió dels seus clients i de les comandes. Per dissenyar el sistema es poden utilitzar els serveis següents (que no es poden modificar):

- ✓ Servei Proveïdor Pizzes (SvJoLaFaig): aquest servei proporciona informació de les pizzes que elabora el proveïdor JoLaFaig. El proveïdor de pizzes pot canviar el preu de les pizzes quan ho consideri convenient.

context SvJoLaFaig::ObtéDadesPizza(nomP:String):
TupleType(de:String,pr:Float)

exc *pizza-no-existeix*: la pizza amb nom *nomP* no existeix

post *result* = obté la descripció i el preu de la pizza

context SvJoLaFaig::EncarregarPizza(nomP:String, units: Integer)

exc *pizza-no-existeix*: la pizza amb nom *nomP* no existeix

exc *units-nok*: *units* <= 0

exc *no-hi-ha-estoc*: no hi ha estoc suficient per servir les *units* de la pizza *nomP*

post *actualitza-estoc:* decrementa l'estoc de la pizza *nomP* en les unitats *units*

- ✓ Servei Empresa Entrega (SvEntrega): cada empresa d'entrega disposa d'un servei per enregistrar les entregues a fer, així com el temps d'entrega, el preu del servei, la seva adreça i el nombre d'entregues fetes. Els noms de les empreses poden usar-se per localitzar els serveis (per exemple, el servei de l'empresa PizzaDelivery es diu SvPizzaDelivery). Les empreses d'entrega poden canviar les seves dades quan vulguin. Les operacions proporcionades pel servei, entre d'altres, són:

context *SvEntrega*::ObtéDadesEmpresa():
TupleType(nomE:Str,te:Float,ps:Float,ad:Str)

post *result* = obté el nom, temps d'entrega, preu del servei i adreça de l'empresa

context *SvEntrega*::Entregar()

post *entregar*: incrementa el *numEntreguesFetes* en una unitat

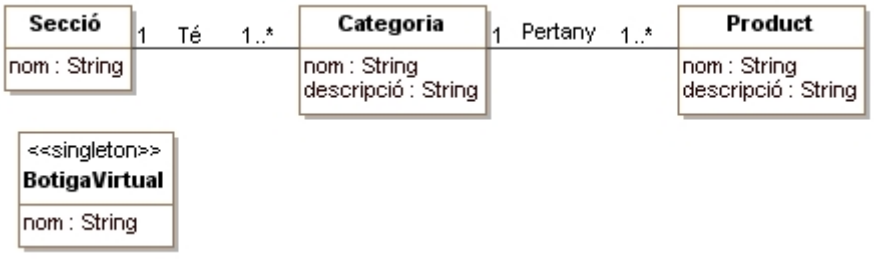
Es demana:

(a) Mostreu com queda el diagrama de classes del sistema de la nostra empresa. Justifiqueu el diagrama obtingut.

(b) Diagrama de seqüència de l'operació FerComanda. Useu Domain Model, Data Mapper i controlador transacció. Suposeu les navegabilitats inicials mostrades a l'esquema. Com a criteri de disseny principal en aquest problema volem minimitzar el nombre d'invocacions remotes. Com a criteri secundari subordinat a l'anterior també es vol minimitzar la redundància de les dades. A continuació disposeu del contracte de l'operació a dissenyar:

```
context CapaDeDomini::FerComanda(dniCl: String, data:date, hora:
Float, pizzes:Set(TupleType(nomP: String, units: Integer)),
nomE:String): Float
pre client-existeix: el client amb dniCl existeix
pre pizzes-existeixen: les pizzes del conjunt pizzes existeixen i
units>0
pre data-hora-correctes: la data i hora dels paràmetres és
correcta
pre empresa-existeix: l'empresa amb nomE existeix
exc comanda-existeix: la comanda existeix
exc no-es-pot-servir: no hi ha estoc suficient per servir les
pizzes de la comanda
post comanda-creada: es crea la comanda i les associacions amb
client i empresa
post item-comanda-creada: es crea un itemComanda per cada tupla
del conjunt pizzes i s'associa amb la comanda i amb la pizza.
post decrementa-estoc: es decrementa l'estoc de les pizzes
demanades a la comanda
post incrementa-numEntregues: s'incrementa numEntreguesFetes de
l'empresa nomE
post result= calcula i retorna el preu de la comanda. Aquest preu
es calcula com: preuComanda= (sumatori del preu de totes les
pizzes de la comanda + el preu del servei d'entrega) * (1 +
percBenefici)
```

8. Volem dissenyar una botiga virtual. Aquesta botiga ven productes que tenen un nom i una descripció. Els productes pertanyen a categories que també tenen el seu nom i la seva descripció. Les categories pertanyen a seccions de la botiga. A continuació disposeu de l'esquema conceptual del sistema:



Restriccions textuals:
RT1. Claus: (Secció, nom); (Categoria, nom); (Producte, nom);

La capa de domini ofereix la següent operació:

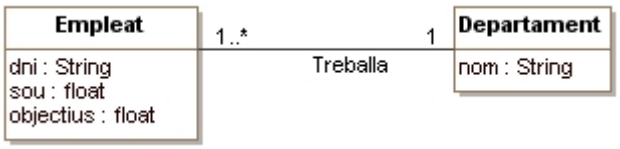
context CapaDeDomini::obtenirInformacióSecció (nomS: String):
Set(TupleType(nomCat:String, desc:String, prods:
Set(TupleType(nom:String, desc:String)))

exc *seccióNoExisteix*: La secció identificada per *nom* no existeix
post result = retorna el nom i la descripció de les categories associades a la secció i per a cada categoria el nom i descripció dels productes de la categoria. Aquesta operació ha d'obtenir la informació ordenada segons el criteri d'ordenació que estigui establert per a la secció. Aquest criteri pot variar al llarg del temps. Els criteris d'ordenació disponibles són: 1) ordre per nom categoria i dins de la categoria per nom de producte o 2) ordre per descripció de la categoria i dins de la categoria per descripció del producte.

Es demana:

(a) Feu el diagrama de seqüència de l'operació anterior. Cal indicar explícitament les comprovacions o aspectes que considereu rellevants i que no apareguin al diagrama de seqüència. No cal definir els algorismes d'ordenació. Podeu suposar l'existència d'una operació d'ordenació
(b) Feu el diagrama de classes de la capa de domini.

9. Una empresa del sector farmacèutic ens ha demanat que li dissenyem un sistema per calcular el sou dels seus empleats. Aquesta empresa està formada per diferents departaments i cada departament té assignats els seus empleats. L'empresa vol poder canviar de forma dinàmica el càlcul dels sous dels seus empleats. En concret vol tenir la possibilitat de calcular el sou de cada empleat de diferents formes i poder variar aquest càlcul per cada empleat en funció de la productivitat i d'altres criteris que estableixi l'empresa. L'empresa estebleix tres formes de fer el càlcul del sou dels seus empleats: sou amb bonus (souEmpleat + bonus definit per l'empresa), sou amb objectius (souEmpleat*2 si l'empleat ha assolit uns objectius superiors a un valor establert per l'empresa i souEmpleat en cas contrari) o sou de crisis (souEmpleat - percentatge definit per l'empresa*souEmpleat). A continuació disposeu de l'esquema conceptual del sistema:



Restriccions textuals:
RT1. Claus: (Empleat, dni); (Departament, nom)

La capa de domini ofereix les següents operacions:

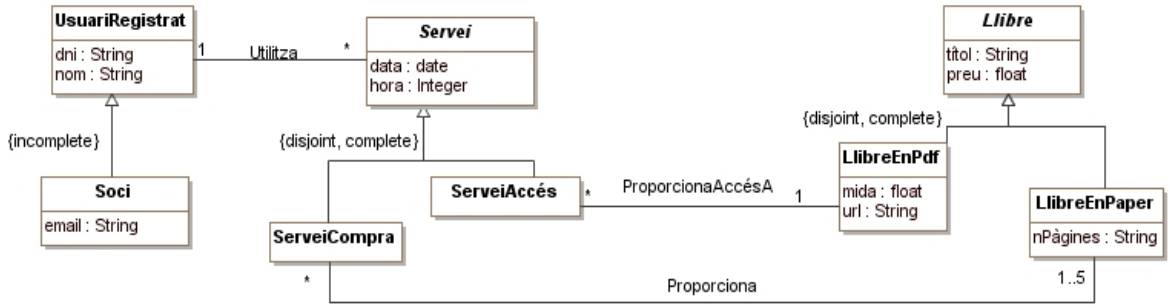
context CapaDeDomini::calcularSou (): Set(dniEmp:String, nomDept:String, sou:float)
exc *noHiHaEmpleats*: no hi ha empleats definits al sistema
post result = retorna el dni, el nom del departament de l'empleat dni i el sou a pagar per tots els empleats de l'empresa. La forma de càlcul del sou de cada empleat estarà definit per l'empresa.

context CapaDeDomini::assignaCàlculSou (dniEmp:String,perc: Integer)
pre *percMésGranQue0*: el percentatge perc és més gran que 0
exc *noExisteixEmpleat*: no existeix l'empleat dniEmp
post *assigna*: s'assigna el càlcul del sou de l'empleat dni a pagament de crisi amb el percentatge perc.

Es demana:

(a) Feu el diagrama de seqüència de les operacions anteriors. Cal indicar explícitament les comprovacions o aspectes que considereu rellevants i que no apareguin al diagrama de seqüència. No cal definir els algorismes d'ordenació. Podeu suposar l'existència d'una operació d'ordenació
(b) Feu el diagrama de classes de la capa de domini.

10. Una llibreria que ven llibres per internet vol dissenyar un sistema software per enregistrar els llibres que ven als seus clients. A continuació disposeu d'un fragment l'esquema conceptual del sistema a dissenyar:



R.I. Textuals:
- Claus: (UsuariRegistrat, dni); (Soci, email); (Servei, UsuariRegistrat:dni + data + hora); (Llibre, títol); (LlibreEnPdf, url)
- Un usuari registrat que no sigui soci només pot comprar com a mínim un i com a màxim tres llibres en paper a cada servei de compra.
- Un usuari registrat que no sigui soci només pot accedir un únic cop a un mateix llibre en pdf.
- Un llibre en pdf es pot accedir com a molt 100 vegades en una data.
- Un llibre pot ser comprat com a molt 3 vegades (en serveis de compres diferents) per un soci.

Considereu el cas d'ús d'especificació *EsborrarServeisAntics* amb un únic esdeveniment amb el mateix nom. Després de fer l'assignació de responsabilitats a les capes disposem del contracte de l'operació de la capa de domini:

context CapaDomini:: *EsborrarServeisAntics* (data: date, preu: float)
pre *dataCorrecta*: la data *data* és vàlida i no és futura.
pre *preuCorrecte*: el preu *preu* és més gran que 0.
exc *noHiHaServeis:* no hi ha serveis al sistema.
exc *noHiHaServeisAntics:* hi ha serveis al sistema però no hi ha serveis amb data anterior a la data indicada al paràmetre que tinguin un preu del servei (mirar definició a la post) inferior al preu indicat en el paràmetre.
post *esborrarServeisAntics*: s'eliminen tots els serveis (i les seves associacions) amb data anterior a la data indicada al paràmetre que tinguin un preu del servei inferior al preu indicat al paràmetre. El preu d'un servei serà un valor calculat per a cada servei. Aquest valor serà el sumatori dels preus dels llibres assignats al servei menys un descompte que s'aplica a cada servei. Podeu suposar que inicialment aquests descomptes seran: 1) una quantitat determinada i 2) un percentatge de descompte. Es vol que més endavant es puguin afegir noves formes de calcular descomptes sempre en funció del preu (amb els mínims canvis possibles). A cada servei se li assignarà la forma de calcular el descompte en el moment de la seva creació.

Suposant que la **navegabilitat de les associacions és doble**, es demana que dissenyeu el diagrama de seqüència de l'operació *EsborrarServeisAntics*. Indiqueu al diagrama de seqüència les classes que són singleton i les interfícies.