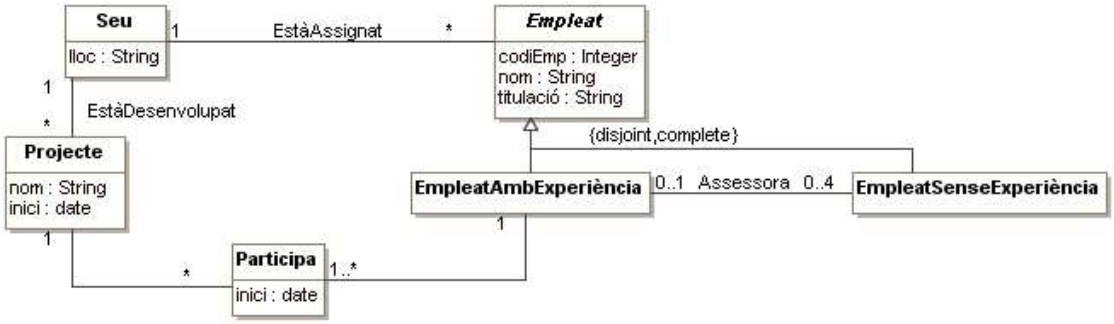
ARQUITECTURA DEL SOFTWARE

Unit 3.2: Domain Layer Design

Enunciats Exercicis

Exercici 1 (Patrons)

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.

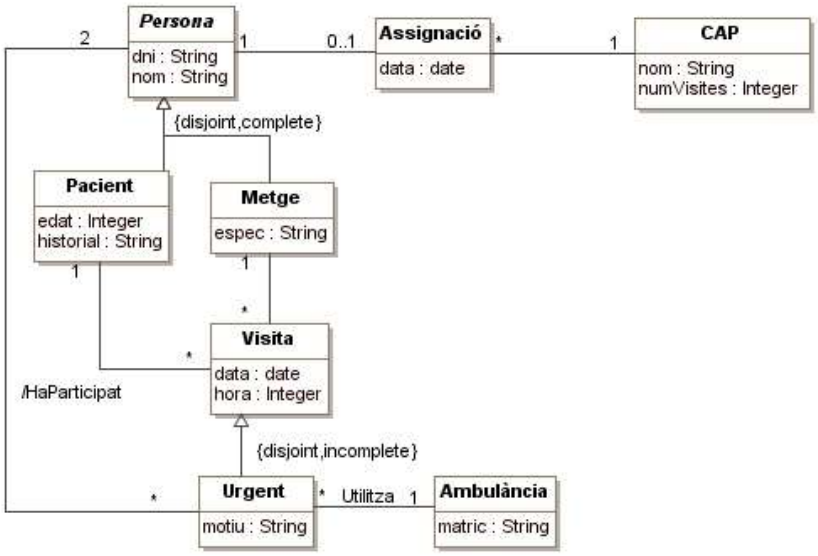
context CapaDeDomini::llistaAss (nomPr: String): Set(String)
exc *projecteNoExisteix*: El projecte identificat per *nomPr* no existeix
post result = retorna els noms dels empleats sense experiència que estan assignats a la seu on s'està desenvolupant el projecte amb *nomPr* i que són assessorats per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

Exercici 2 (Patrons)

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 del diagrama de classes de la capa de domini:



R.I. Textuals:
- Claus classes no associatives: (Persona, dni); (CAP, nom); (Hora, data+hora); (Ambulància, matric)
- No poden haver-hi dues visites amb el mateix pacient, data i hora.
- No poden haver-hi dues visites amb el mateix metge, data i hora.
- El numVisites d'un CAP ha de coincidir amb el nombre de visites que tenen els pacients assignats a aquest CAP.
- 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.

Contracte de l'operació anul.lació de la capa de domini (assumim que HaParticipat s'ha materialitzat):
context CapaDomini::anul.lació (dniPac: String, data: Date, hora: Integer)

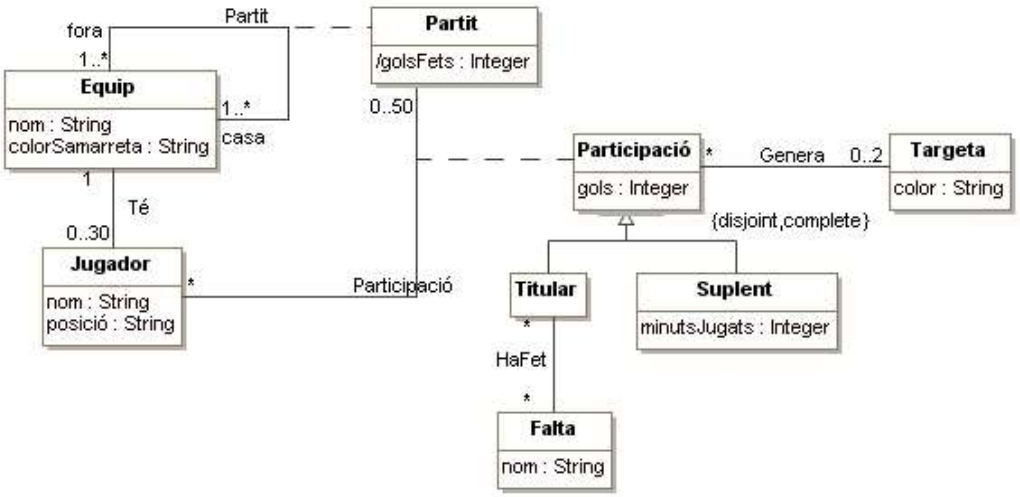
exc *visitaNoExisteix*: la visita no existeix
exc *visitaUrgentAmbProblemaCor*: la visita és urgent i té com a motiu "Problema de Cor"
exc *visitesMínimes*: el CAP té 16 visites
post *eliminaVisita*: dóna de baixa l'ocurrència de la classe Visita i les seves associacions
post *eliminaUtiliza*: si la visita és urgent es dóna de baixa l'associació Utilitza
post *decrementaNumVisites*: es decrementa el numVisites del CAP
post *eliminaHaParticipat*: si és urgent es dóna de baixa l'associació HaParticipat

Considereu que les navegabilitats de *Utilitza* i *HaParticipat* és doble. Es demana:
(a) Feu el diagrama de seqüència de l'operació anul·lació de la capa de domini. Cal indicar explícitament les comprovacions o aspectes que considereu rellevants i que no apareguin al diagrama de seqüència.
(b) Feu el diagrama de classes de la capa de domini.
(c) Suposeu ara que decidim que l'associació *HaParticipat* es calcula. Feu els diagrames de seqüència de les operacions resultants del càlcul.

Exercici 3 (Patrons)

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:

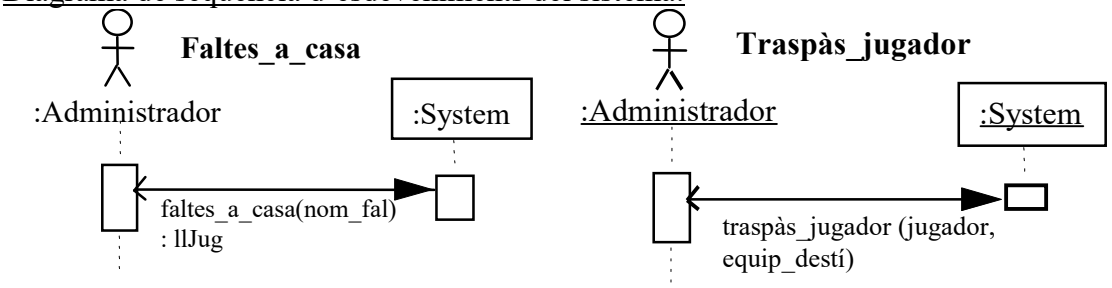


R.I Textuals:

- 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.

Exercici 4 (Serveis)

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:

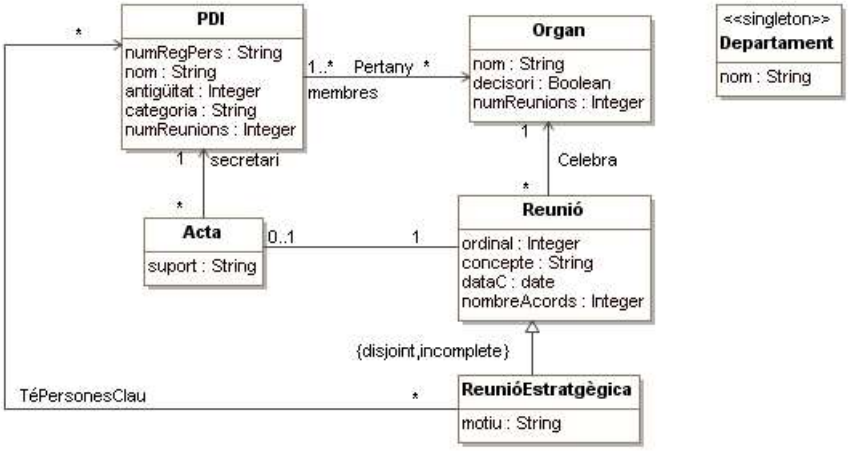
context Viatge::calculaPreu(): Float
post 2.1: retorna el preu del viatge en euros

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.

Exercici 5 (Serveis)

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

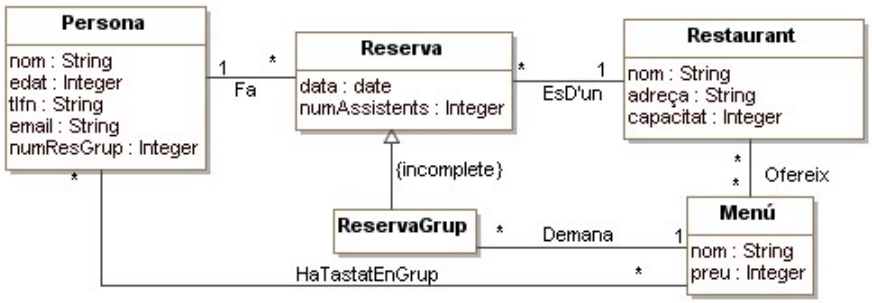
Es demana:
(a) Diagrama de classes del disseny del sistema d'informació incloent-hi els atributs. No cal declarar les operacions auxiliars del sistema. Indiqueu-ne els acoblaments i navegabilitats finals.
(b) Diagrama de seqüència de les dues operacions de la capa de domini del sistema, fent servir Domain 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 DirectoriUPC 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.

Exercici 6 (Serveis)

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*

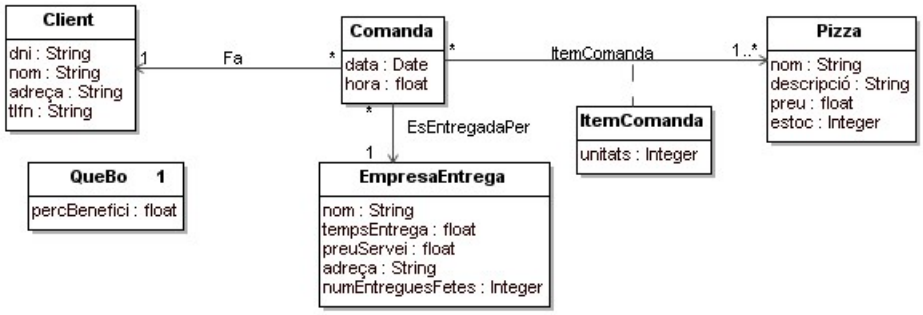
context SvGestorReserves::obtenirOcupacióxData(inout dOcup: DTOOcupacióReserva)
post *dOcup.numAssistents*= sumatori dels assistents de les reserves del restaurant *dOcup.nomRes* en data *dOcup.dt*.
**La classe DTOReserva té com atributs nom persona, nom restaurant, data i número d'assistents.*
**La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

Exercici 7 (Serveis)

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:

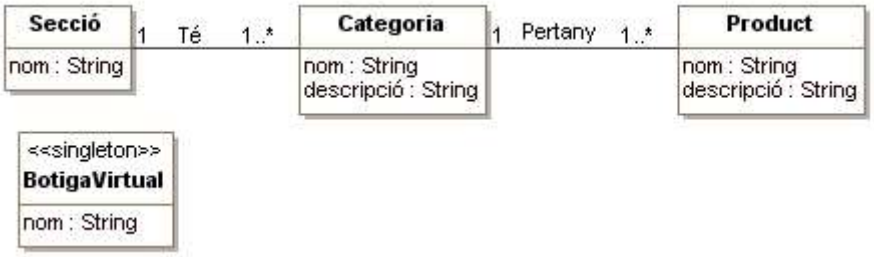
(a) Mostreu com queda el diagrama de classes del sistema de la nostra empresa. Justifiqueu el diagrama obtingut.

(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)
```

Exercici 8 (Patrons)

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:

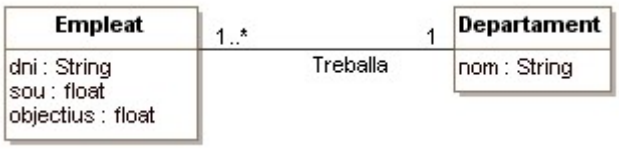
context CapaDeDomini::obtenirInformacióSecció (nomS: String):
Set(TupleType(nomCat:String, desc:String, prods: Set(TupleType(nom:String, desc:String)))
exc *seccióNoExisteix*: La secció identificada per *nom* no existeix
post result = retorna el nom i la descripció de les categories associades a la secció 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.

Exercici 9 (Patrons)

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:

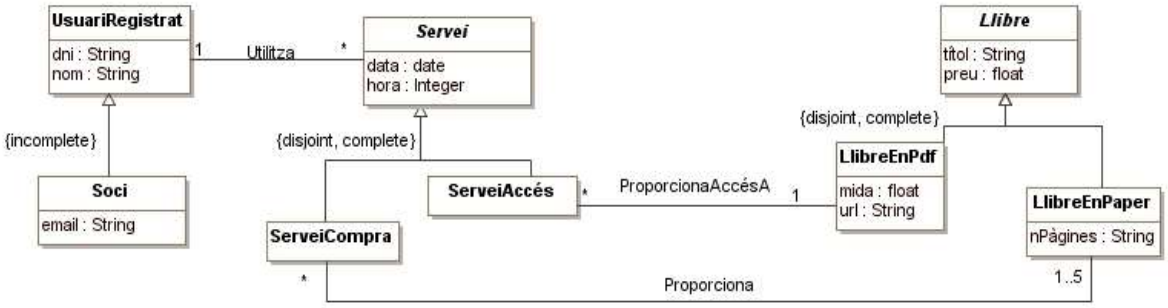
context CapaDeDomini::calcularSou (): Set(dniEmp:String, nomDept:String, sou:float)
exc *noHiHaEmpleats*: no hi ha empleats definits al sistema
post result = retorna el dni, el nom del departament de l'empleat dni i el sou a pagar per tots els empleats de l'empresa. La forma de càlcul del sou de cada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

Exercici 10 (Patrons)

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.

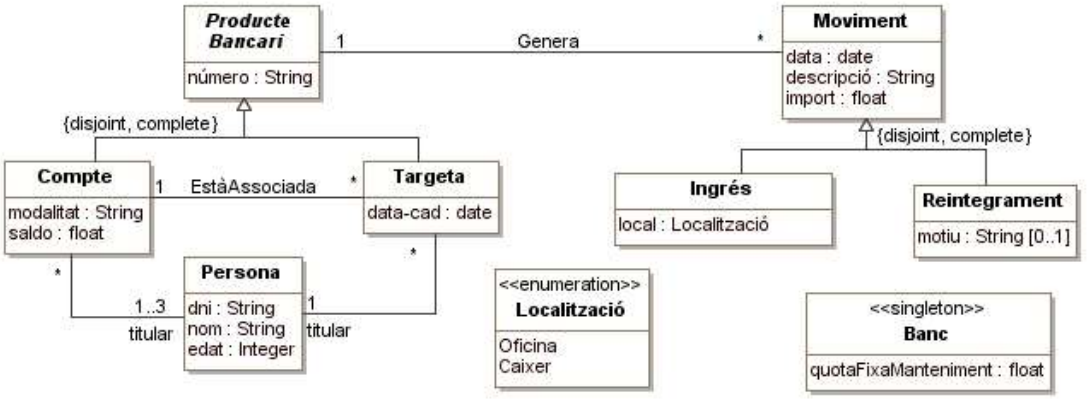
Considereu el cas d'ús d'especificació *EsborrarServeisAntics* amb un únic esdeveniment amb el mateix nom. Després de fer l'assignació de responsabilitats a les capes disposem del contracte de l'operació de la capa de domini:

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.

a) 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.

Exercici 11 (Patrons)

Un banc ens ha demanat que dissenyem una part del sistema software que permeti gestionar les comissions de manteniment que cobren als seus clients pels productes bancaris. Aquests productes tenen un número que els identifica i poden ser de dos tipus: comptes o targetes. Dels comptes s'enregistra la modalitat, el saldo i els titulars. De les targetes s'enregistra la data de caducitat, el titular i el compte al que està associada. Els productes bancaris generen moviments en una data, per un import i amb una descripció. Els moviments poden ser ingressos o reintegraments. Dels ingressos sabem la seva localització i dels reintegraments sabem de forma opcional el seu motiu. A continuació disposeu d'un fragment de l'esquema conceptual del sistema a dissenyar:



R.I. Textuals:
- Claus: (Persona, dni); (Producte Bancari, número)
- Tots els integers i floats excepte el saldo han de ser positius.
- No es poden fer moviments amb targetes caducades.
- El titular d'una targeta ha de ser un dels titulars del compte associat.
- Tots els reintegraments amb un import superior a 1000 euros han de tenir el motiu del reintegrament.
- El saldo d'un compte bancari és el sumatori de tots els imports dels ingressos del compte i de totes les targetes associades menys el sumatori de tots els imports dels reintegraments del compte i de totes les targetes associades.
- Altres restriccions no rellevants pel problema

El banc ens demana que dissenyem una funcionalitat per calcular i cobrar la comissió de manteniment anual a cada producte bancari. Hi ha dues formes de calcular aquesta comissió: estàndard o reduïda. La forma de calcular aquesta comissió s'assigna al producte bancari quan aquest es crea. La comissió reduïda s'aplica a aquells productes bancaris on els seus titulars compleixen unes condicions (que no enregistrem en el sistema) definides pel banc (per exemple, jubilats, aturats, etc...). A la resta de productes se'ls aplicarà una comissió estàndard. El banc pot canviar aquestes condicions quan ho cregui oportú. També pot afegir noves fórmules de calcular les comissions en el futur (per exemple, comissió super-reduïda) que sempre dependran del (nombre d'ingressos en caixers + nombre de reintegraments amb motiu anuals). A més, els productes també poden canviar el tipus de comissió a aplicar en qualsevol moment si els seus titulars varien les seves condicions. Es vol que aquests canvis es puguin fer en temps d'execució. El càlcul de les comissions serà el següent:

- Comissió estàndard = quotaFixaManteniment + (Número ingressos en caixer de l'any + Número reintegraments de l'any amb motiu) * 0,10 + recàrrec.
- Comissió reduïda = (Número ingressos en caixer de l'any + Número reintegraments de l'any amb motiu) * 0,05 - bonificació.

Nota: Tant la bonificació com el recàrrec poden ser diferents per a cada producte bancari.

Considereu el contracte de l'operació de la capa de domini que calcula i cobra la comissió de manteniment anual a cada producte bancari:

context CapaDomini:: *ComissióAnual* (): Set(TupleType (número: String, titulars: Set(nom:String), comissió:Float))

pre *targetesActives*: totes les targetes del sistema estan vigents (no caducades).

exc *noHiHaProductesBancaris*: no hi ha productes bancaris definits al sistema.

post *calculaComissióAnual*: per a cada producte bancari es calcula la comissió per a l'any actual en funció del càlcul de comissió que tingui assignada en el moment de la invocació d'aquesta operació.

post *cobraComissióAnual*: per a cada producte bancari es genera un moviment de tipus reintegrament amb la data actual, la descripció "Comissió Manteniment Anual" i la comissió calculada a l'anterior postcondició. Si la comissió supera els 1000 euros, el motiu del reintegrament ha de ser "Comissió Entitat Bancaria". El moviment estarà associat al producte bancari pel qual s'ha calculat la comissió.

post *decrementaSaldo*: si el producte bancari és un compte es decrementa l'import de la comissió del saldo del compte. Si és una targeta es decrementa l'import de la comissió del saldo del compte associat a la targeta.

post *result* = per a cada producte bancari es retorna el número, el nom dels titulars i la comissió cobrada.

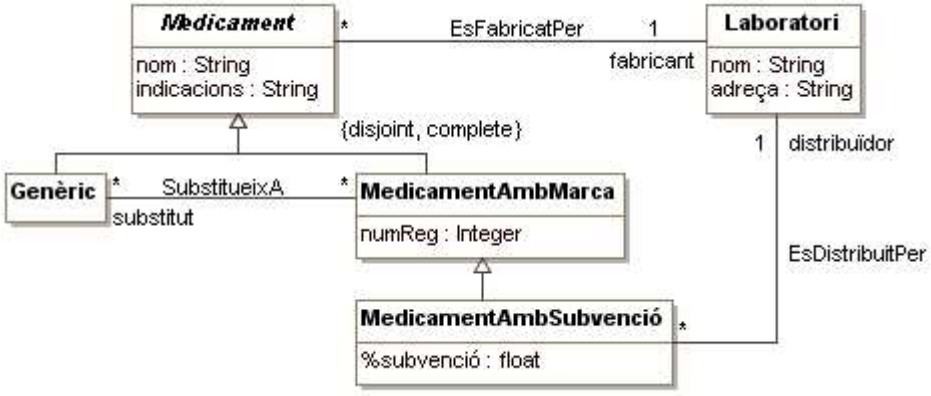
Notes:

- Podeu invocar des d'on ho necessiteu l'operació *getDataActual():data* per obtenir la data actual, *getAnyActual():Integer* per obtenir l'any actual i *getAnyData(d:date):Integer* per obtenir l'any d'una data.
- Per simplificar el problema, no us preocupeu si el saldo d'algun compte queda en negatiu en carregar la comissió anual.

Es demana:
1. Diagrama de seqüència de l'operació *ComissióAnual*. Indiqueu clarament en el diagrama de seqüència els singleton i les interfaces. Considereu en el vostre disseny que:
a. El càlcul la comissió d'un producte bancari també s'ha d'utilitzar en altres casos d'ús (per exemple, per fer simulacions de les comissions a cobrar).
b. La generació de reintegraments d'un producte bancari també s'ha d'utilitzar en altres casos d'ús (per exemple, per fer reintegraments amb una targeta en un caixer).
2. Diagrama de classes de disseny de la capa de domini del sistema. Indiqueu les operacions (a excepció dels getters i els setters) i dieu si són abstractes. Indiqueu també les navegabilitats. No podeu afegir noves classes si no són derivades de l'aplicació de patrons.

Exercici 12

Una agrupació de laboratoris farmacèutics ens ha demanat que dissenyem una part del seu sistema informàtic per gestionar els medicaments que serveix a les farmàcies. Els laboratoris d'aquesta agrupació s'encarreguen de fabricar i distribuir medicaments. Els medicaments que fabriquen poden ser genèrics o amb marca. Cada medicament genèric pot ser substituït per medicaments amb marca i al contrari. Alguns medicaments amb marca són subvencionats per la seguretat social. Aquests últims són distribuïts per laboratoris de l'agrupació (que poden ser diferents del laboratori de fabricació). A continuació disposeu de l'esquema conceptual i dels contractes de les operacions a dissenyar.

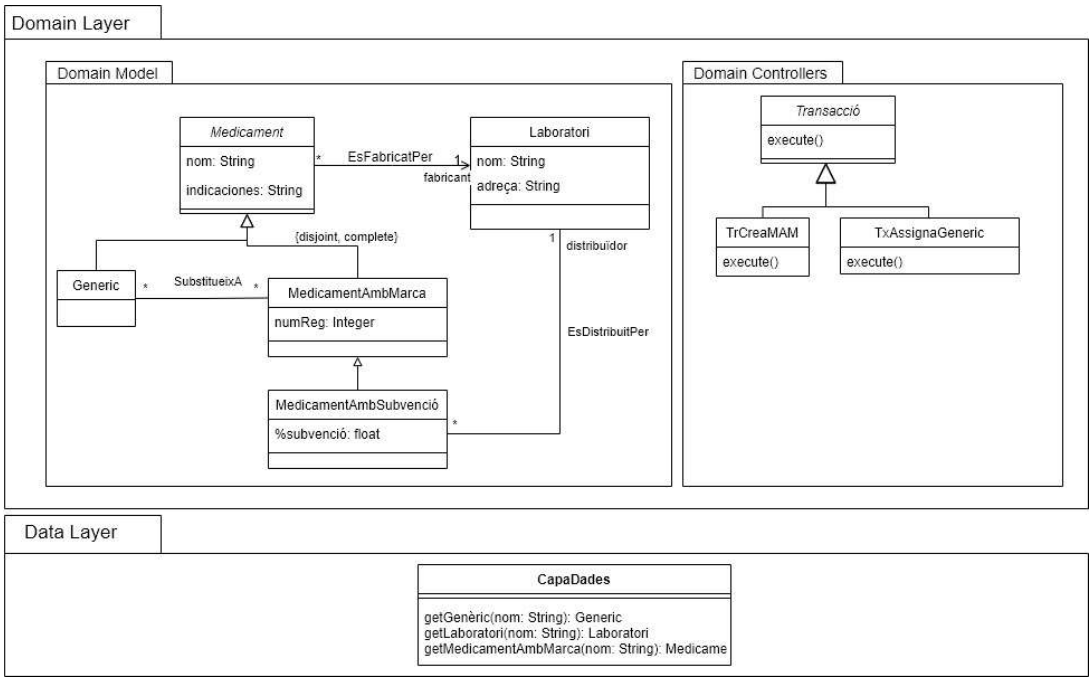


R.Integritat Textuals:
- Claus classes no associatives: (Medicament, nom); (Laboratori, nom)
- Els genèrics d'un medicament amb marca no poden ser fabricats pel mateix laboratori que fabrica el medicament amb marca.

context CapaDomini::CreaMedicamentAmbMarca (nomMed:String, indicacions: String, numReg: Integer, nomLab:String, gen:Set(nom:String)
exc *genNoExisteix*: algun genèric de *gen* no existeix.
exc *medicamentAmbMarcaExisteix*: el medicament amb *nomMed* ja existeix.
exc *laboNoExisteix*: el laboratori amb nom *nomLab* no existeix.
exc *mateixLabo*: algun genèric de *gen* té el laboratori de fabricació *nomLab*.
post creaMedicamentAmbMarca: es crea el medicament amb marca (el valor dels paràmetres s'assignen als atributs), s'associa el medicament al laboratori de fabricació *nomLab* i als genèrics del conjunt *gen*.

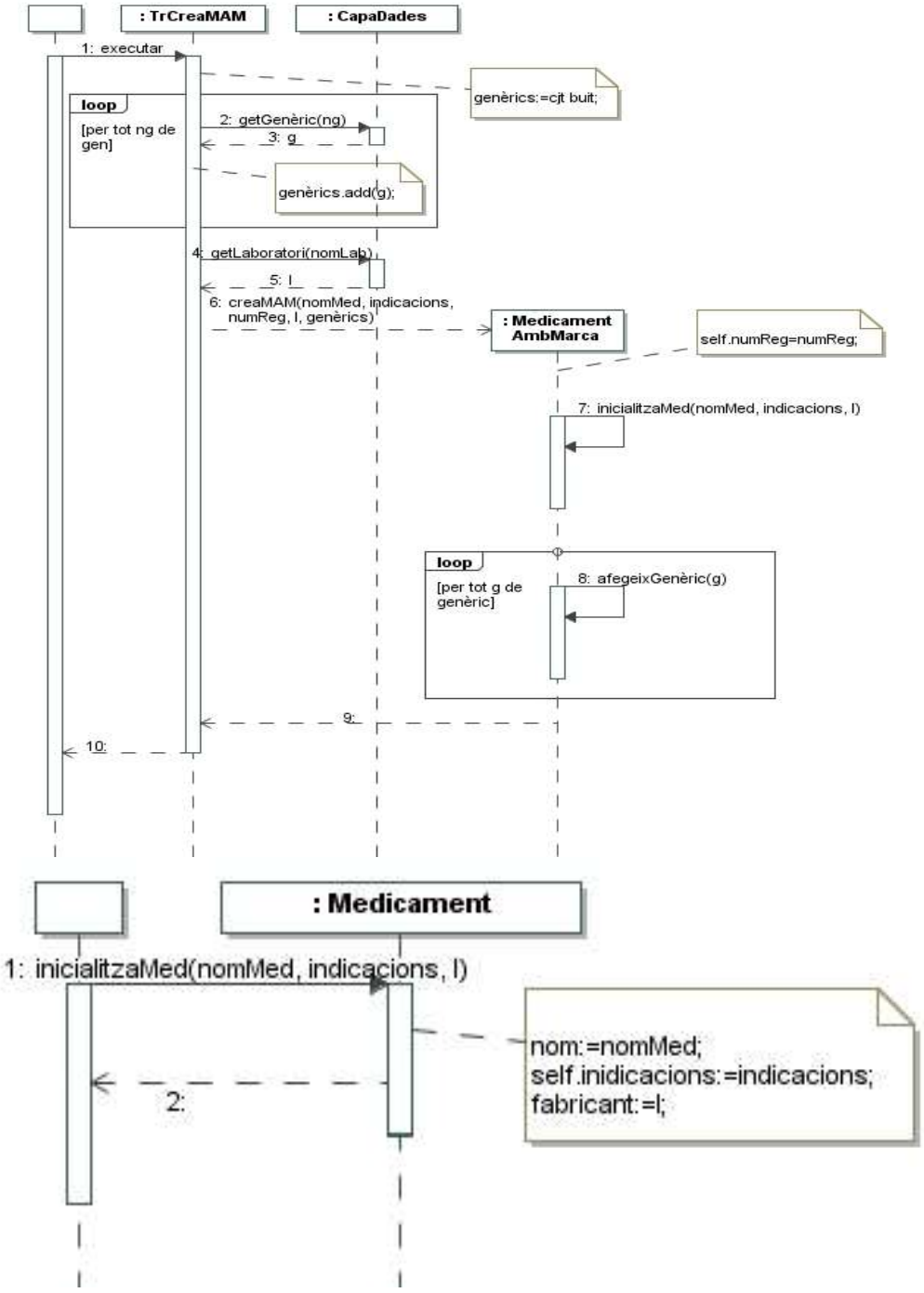
context CapaDomini::AssignaGenèric (nomMed:String, nomGen: String)
exc *medicamentAmbMarcaNoExisteix*: el medicament amb marca amb nom *nomMed* no existeix.
exc *genèricNoExisteix*: el genèric amb nom *nomGen* no existeix.
exc *jaTéGenèric*: el medicament amb marca amb nom *nomMed* ja té el genèric amb nom *nomGen*.
exc *mateixLabo*: el medicament amb marca amb nom *nomMed* i el genèric amb el nom *nomGen* tenen el mateix laboratori de fabricació.
post assignaGenèric: s'associa el genèric al medicament amb marca.

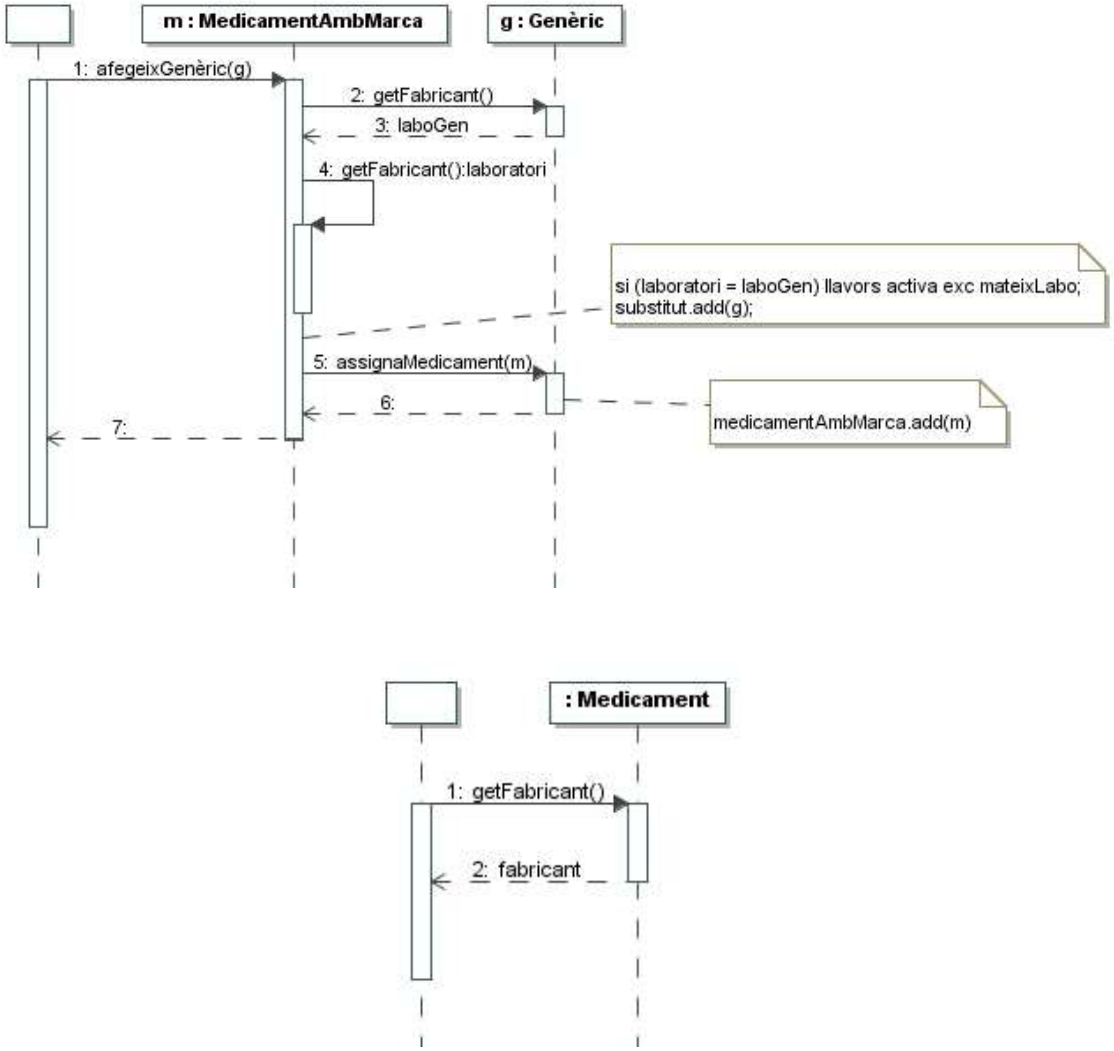
El diagrama de classes que hem proposat al dissenyar aquest sistema es el següent:



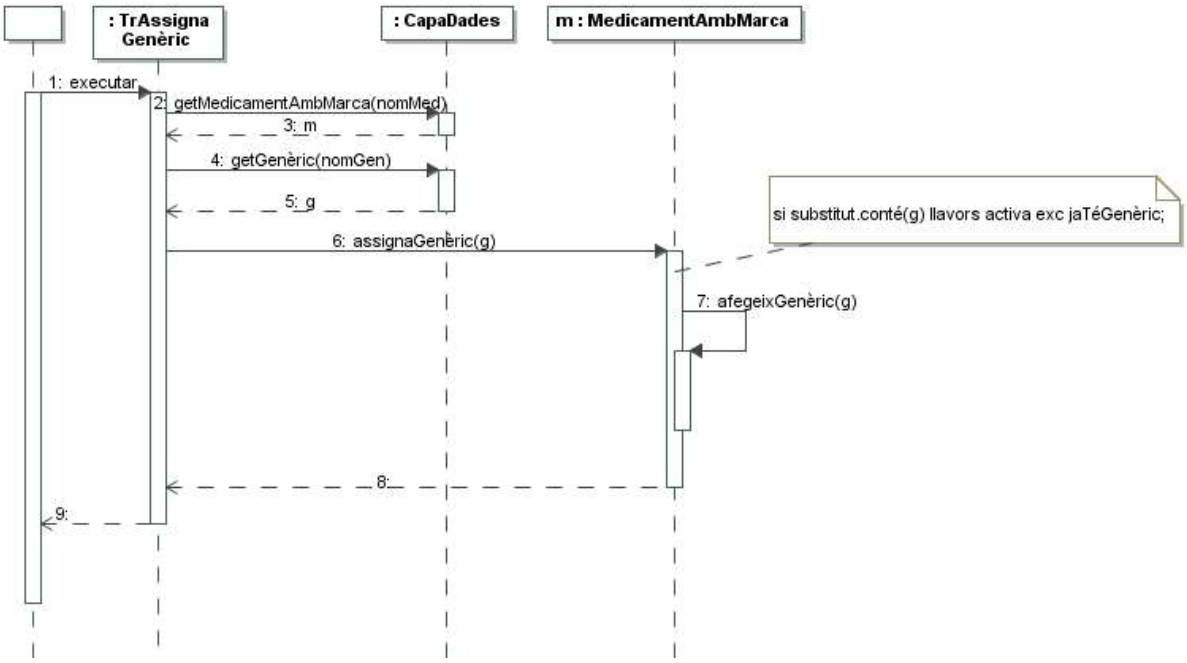
I els diagrames de seqüencia dels controladors transacció que s'encarregaran de l'execució dels 2 contractes anteriors són:

(1) Diagrama de seqüència de la operació CreaMedicamentAmbMarca


: TrCreaMAM
: CapaDades
1: executar
genèrics:=cjt buit;
loop
[per tot ng de gen]
2: getGenèric(ng)
3: g
genèrics.add(g);
4: getLaboratori(nomLab)
5: l
6: creaMAM(nomMed, indicacions, numReg, l, genèrics)
: Medicament AmbMarca
self.numReg=numReg;
7: inicialitzaMed(nomMed, indicacions, l)
loop
[per tot g de genèric]
8: afegeixGenèric(g)
9:
10:
: Medicament
1: inicialitzaMed(nomMed, indicacions, l)
nom:=nomMed;
self.inidicacions:=indicacions;
fabricant:=l;
2:



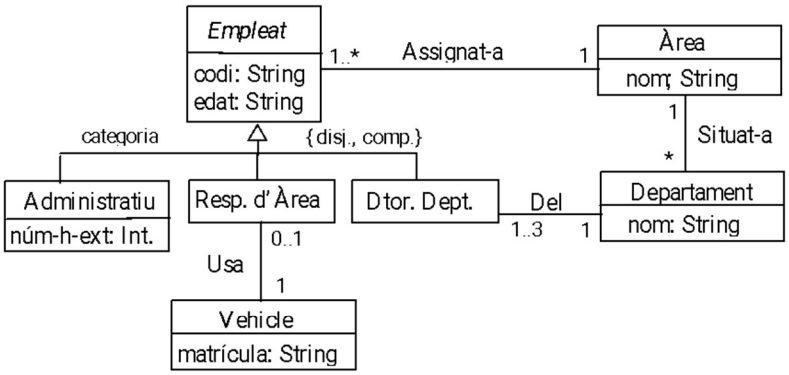
(2) Diagrama de seqüència de la operació AssignaGenèric



Donat el nostre disseny per les dues operacions, es demana:
a) Identificar problemes en el disseny proposat pels principis de disseny següents: Acyclic Dependencies Principle (ADP) (explicat a Unit 2.1- Layered Architecture), Dependency Inversion Principle (DIP) (explicat a Unit 2.2- Object Oriented Architecture).
b) Proposar un disseny alternatiu millorant els problemes trobats a l'apartat a. Haureu d'aplicar patrons de disseny quan sigui necessary.

Exercici 13 (Patrons)

Una empresa ens ha demanat que li dissenyem un sistema software per mantenir la informació de la categoria laboral dels seus empleats. Els empleats de l'empresa s'identifiquen per codi i estan assignats a una àrea geogràfica. Un empleat té una categoria laboral que pot ser: administratiu, responsable d'àrea o director de departament. Un empleat es pot contractar en qualsevol categoria però una vegada contractat només pot ascendir a la categoria superior (l'ordre de les categories és administratiu, responsable d'àrea i director de departament. Dels empleats que són administratius se'n guarda el nombre d'hores extres que han fet en un mes. Dels responsables d'àrea, se'n coneix el vehicle que poden utilitzar en els seus desplaçaments. Finalment, dels directors de departament cal enregistrar el departament que dirigeixen. A continuació disposeu de l'esquema conceptual del sistema:



R.I. Textuals:
- Claus de les classes no associatives: (Empleat, codi); (Àrea, nom); (Vehicle, matrícula); (Departament, nom).
- Un administratiu no pot fer més de 20 hores extres.
- No pot ser que hi hagi una àrea que no tingui assignat cap responsable d'àrea.
- Els directors de departament no poden tenir més de 30 anys.
- L'àrea d'assignació del director d'un departament ha de coincidir amb l'àrea on està situat el departament.

Considereu el cas d'ús d'especificació fer-director amb un únic esdeveniment amb el mateix nom.

context CapaDomini:: fer-director (codi-emp: String, nom-dept: String)
pre *empleatExisteix*: l'empleat amb codi-emp existeix.
pre *departamentExisteix*: el departament amb nom-dept existeix.
exc *empleatNoResp:* l'empleat amb codi-emp no és responsable d'àrea.
exc *massaDirectors:* el departament nom-dept ja té 3 directors.

exc *senseÀrea:* l'empleat és l'únic responsable d'àrea de l'àrea al que està assignat.
exc *massaGran:* l'empleat té més de 30 anys.
exc *diferentsÀrees:* l'àrea del departament nom-dept i la de l'empleat codi-emp són diferents.
post *ascendir*: l'empleat especificat passa a ser director de departament, del departament identificat per nom-dept. S'elimina l'associació Usa entre el responsable d'àrea i el vehicle. Es crea la instància de l'associació Del entre Departament i Dtor. Dept.

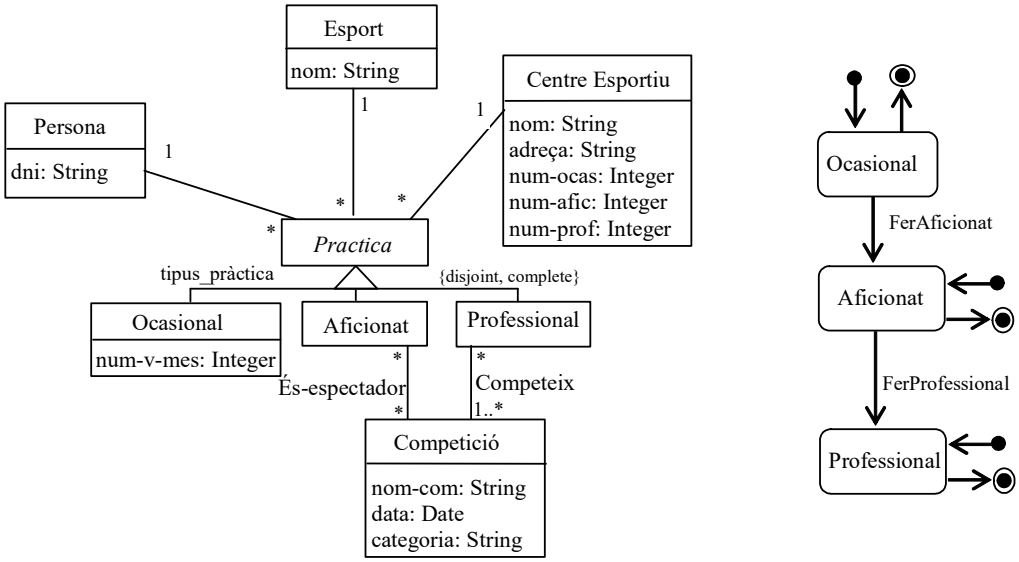
Es demana:

a) Diagrama de seqüència de l'operació del sistema *fer-director* , suposant que totes les navegabilitats són dobles. Cal que indiqueu amb comentaris tot el que no aparegui de forma explícita en els diagrames de seqüència.

b) Diagrama de classes on es vegi l'aplicació del patró estat. No cal afegir les operacions.

Exercici 14 (Patrons)

Una associació de centres esportius ens ha demanat que li dissenyem un sistema software per mantenir la informació de les persones que practiquen esports en els seus centres. Les persones poden practicar els esports de forma ocasional, com aficionats o com professionals. Les persones s'identifiquen per dni i els esports pel seu nom. Dels centres esportius es guarda el nom, l'adreça i el nombre de persones que practiquen esport de forma ocasional, com aficionats i com professionals. De les persones que practiquen l'esport ocasionalment se'n guarda el nombre de vegades que hi van a fer esport en un mes. Dels aficionats, se'n coneix les competicions en les que són espectadors. Finalment, dels professionals cal enregistrar les competicions en les que competeixen. Les competicions s'identifiquen pel seu nom. A més també es guarda de les competicions la data de la celebració i la categoria. A continuació disposeu de l'esquema conceptual del sistema:



R.I. Textual:

- Claus classes no associatives: (Persona, dni); (Esport, nom); (Centre Esportiu, nom); (Competició, nom-com).
- No poden existir dues pràctiques per a la mateixa persona i esport.
- Una persona que practica un esport ocasionalment no pot anar més de quatre vegades al mes al centre esportiu.
- Una persona que practica un esport de forma professional no pot competir en competicions de categoria 'Aficionat'.
- De totes les pràctiques que té una persona, només una pot ser professional.
- En un centre esportiu es poden practicar com a molt 20 esports diferents.
- L'atribut num-ocas és igual al nombre de persones que practiquen esport en el centre de forma ocasional.
- L'atribut num-afic és igual al nombre de persones que practiquen esport en el centre de forma aficionada.
- L'atribut num-prof és igual al nombre de persones que practiquen esport en el centre de forma professional.

Considereu el cas d'ús d'especificació *FerProfessional* 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:

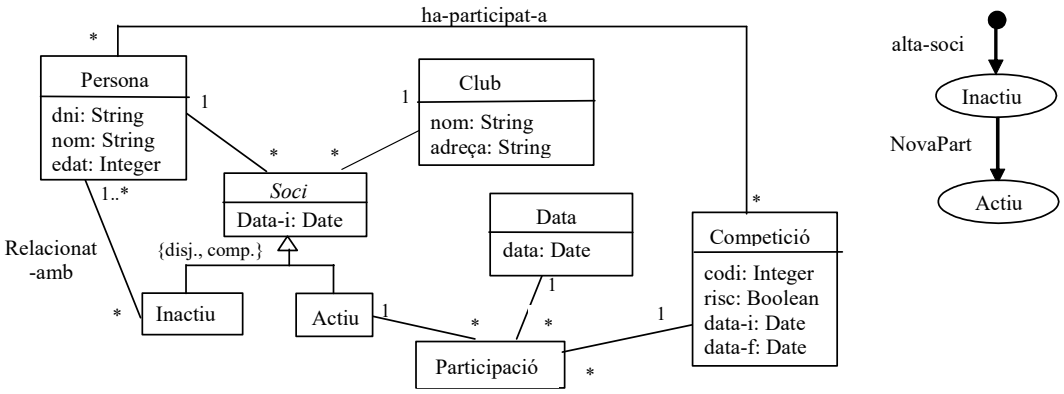
context CapaDomini:: *FerProfessional* (dni: String, nom-e: String, nom-com:String)
pre *competicióExisteix*: la competició amb nom nom-com existeix.
pre *personaExisteix*: la persona amb dni existeix.
pre *esportExisteix*: l'esport amb nom-e existeix.
exc *compd'Aficionat:* la competició nom-comp és de categoria 'Aficionat'.
exc *noAficionat:* la persona amb dni no practica l'esport amb nom-e com aficionat.
exc *personaJaTéPràcticaProfessional:* la persona amb dni ja té una practica com a professional.
post *passaAProfessional*: la persona amb dni passa a practicar l'esport nom-e de forma professional. S'elimina l' associació És_espectador entre l'aficionat i les competicions. Es crea l'associació Competeix entre el professional i la competició amb nom_com. S'incrementa en 1 l'atribut num-prof del club esportiu de la pràctica i es decrementa en 1 l'atribut num-afic. S'eliminen les instàncies de l'associació Participa entre la persona i les competicions en les que era espectador com aficionat. Es crea la instància de l'associació Participa entre la persona i la competició nom-comp.

Es demana:

a) Diagrama de seqüència de l'operació *FerProfessional*, suposant que la navegabilitat de les associacions Competeix i És_espectador és doble. Cal que indiqueu amb comentaris tot el que no aparegui de forma explícita en els diagrames de seqüència.
b) Indiqueu la navegabilitat de la vostra solució (només paquet domain model).

Exercici 15 (Patrons)

Una agrupació de clubs esportius ens ha demanat que li dissenyem una part d'un sistema software per gestionar les participacions dels seus associats a les competicions que organitza. Les persones s'identifiquen per dni i se n'enregistra també el seu nom i edat. Els clubs esportius s'identifiquen per nom i s'enregistra també la seva adreça. També s'enregistra informació de les persones que estan associades als clubs esportius, és a dir, dels socis dels clubs.
Un soci d'un club esportiu pot ser inactiu o actiu. Pels socis inactius s'enregistren les persones amb qui tenen algun tipus de relació (siguin o no del mateix club esportiu). Dels socis actius, s'enregistren els dies en què han participat a una determinada competició. Les competicions s'identifiquen per codi i se sap també si són de risc o no, la seva data d'inici i la data de finalització. Un soci pot participar a una mateixa competició tants cops com vulgui. A continuació disposeu de l'esquema conceptual del sistema:



R.I. Textuals:
- Claus classes no associatives: (Persona, dni); (Club, nom); (Data, data); (Competició, codi)
- No poden existir dos socis per a la mateixa persona i club.
- No poden existir dues participacions pel mateix soci actiu i data.
- Una persona menor de 18 anys no pot participar a competicions de risc
- La data de participació d'un soci actiu a una competició ha d'estar entre la data d'inici i de final d'aquesta.
- Una persona no pot participar a una competició com a soci d'un club en una data anterior a la data d'inici de soci del club.
- L'associació *ha-participat-a* relaciona una persona amb les competicions en què ha participat i viceversa.

Considereu el cas d'ús d'especificació *ParticipantsRisc* amb un únic esdeveniment amb el mateix nom i el cas d'ús *NovaParticipació* amb un únic esdeveniment amb el mateix nom.

context CapaDomini:: ParticipantsRisc (dni: String, nom: String): Set(nomP: String)
pre *clubExisteix*: el club amb nom existeix.
exc *sociNoInactiu:* la persona dni no és un soci inactiu del club nom.
post *result* = llista que conté el nom de totes les persones que estan relacionades amb aquest soci i que són socis actius del club nom amb alguna participació a una competició de risc.

context CapaDomini:: NovaParticipació (dni: String, nom: String, codi: String, data: Date)
pre *dataOk*: data correcta, futura i està entre la data d'inici i de fi de la competició amb codi.
exc *noSociClub:* la persona dni no és soci del club nom.
exc *competicióNoExisteix:* la competició codi no existeix.
exc *personaJaParticipa:* el soci ja participa a la competició en aquella data.
post *canviEstat*: Si el soci és inactiu, es dóna de baixa l'associació Relacionat-amb i el soci passa a estar actiu. En qualsevol cas, es dóna d'alta una nova instància de participació definida pel soci actiu, la data i la competició. A més, es dona d'alta una instància de l'associació ha-participat-a entre la persona i la competició.

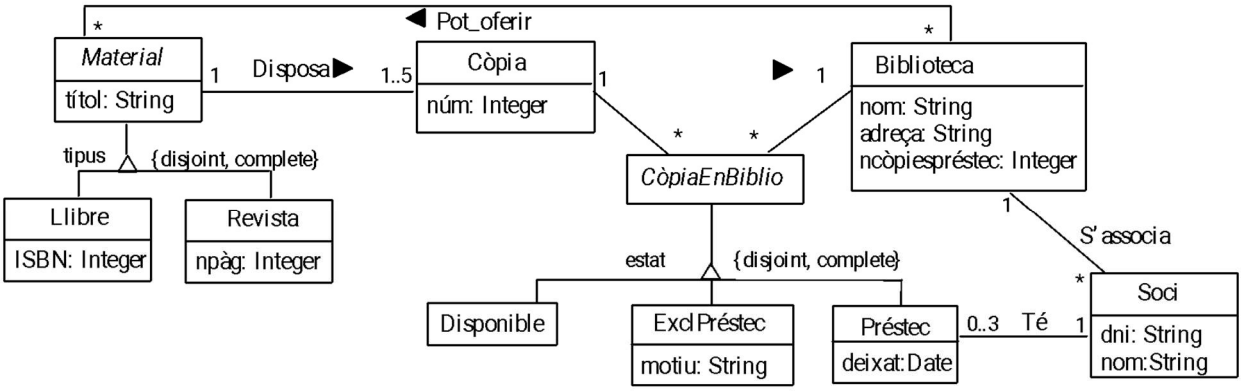
Es demana:

a) Diagrames de seqüència de les operacions *ParticipantsRisc* i *NovaParticipació,* suposant la navegabilitat doble de l'associació *ha-participat-a*.

b) Diagrama de classes resultant (només paquet domain model). No cal posar les operacions.

Exercici 16 (Patrons)

El Consorci de Biblioteques Universitàries de Catalunya (CBUC) ens ha demanat dissenyar un sistema per gestionar el préstec de materials que fan les seves biblioteques. Els materials, identificats pel seu títol, de què disposen les biblioteques són llibres i revistes. Dels llibres en coneixem l'ISBN i de les revistes el número de pàgines. Cada material disposa d'un conjunt de còpies. De cada còpia en sabem el seu número. Les biblioteques, identificades pel seu nom, poden tenir un conjunt de socis que són els que poden utilitzar els seus serveis. Cada còpia de material de què disposa una biblioteca pot estar disponible, exclosa de préstec o en préstec. Si la còpia està exclosa de préstec en sabem el motiu i si la còpia està en préstec en sabem el soci que la té i la data en la que s'ha fet el préstec. Quan una còpia està disponible es pot prestar o excloure de prèstec. Quan està exclosa de prèstec es pot tornar a posar disponible i quan està prestada es pot posar a disponible quan es retorna el prèstec. A continuació disposeu de l'esquema conceptual del sistema:



R.I. Textuals:

Claus de classes no assoc.: (Material, títol); (Biblioteca, nom); (Soci, dni).
No poden existir dues còpies en biblio per a la mateixa còpia i biblioteca.
Un material no pot tenir dues còpies amb el mateix número.
No poden existir dos llibres amb el mateix ISBN.
Un soci només pot tenir en préstec còpies de materials de la biblioteca de la qual és soci.
Un soci no pot tenir dos préstecs del mateix material.
L'atribut ncòpiesenpréstec té el nombre de còpies que la biblioteca té en préstec

Considereu el cas d'ús d'especificació obt_dades_préstec amb un únic esdeveniment amb el mateix nom i el cas d'ús préstec amb un únic esdeveniment amb el mateix nom.

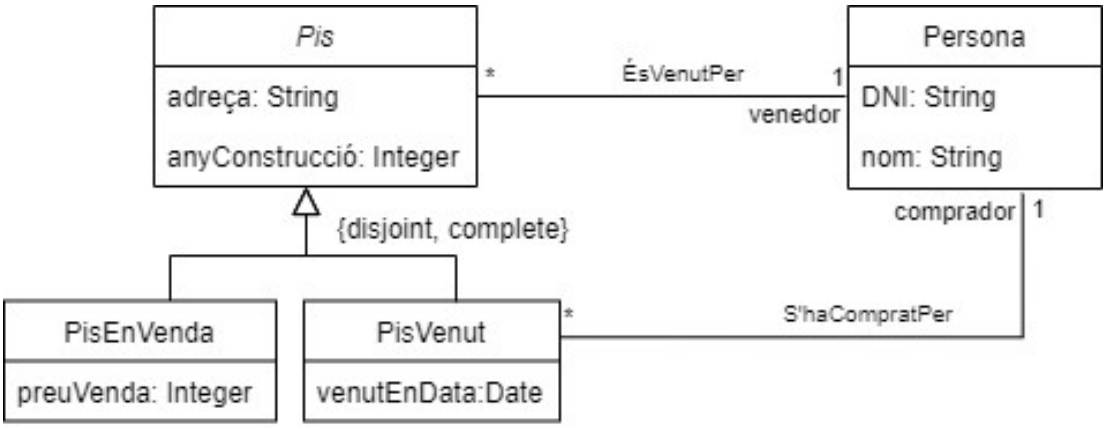
context CapaDomini:: obt_dades_préstec (dni: String): Set(TupleType(títol: String, ISBN o npàg: Integer))
exc *sociNoExisteix:* el soci amb dni no existeix.
post *result* = llista de materials que té en préstec el soci amb dni. Per cada material es proporciona el títol i l'ISBN si és un llibre o el títol i el npàg si és una revista.

context CapaDomini:: préstec (títol: String, dni: String, nom: String)
exc *sociNoExisteix:* el soci amb dni no existeix.
exc *sociNoBiblio:* el soci amb dni no és de la biblioteca nom.
exc *noDisponible:* no existeix una còpia del material disponible a la biblioteca del soci.
exc *jaElTé:* el soci ja té en préstec un material amb el títol de la seva biblioteca.
exc *jaTé3Préstecs:* el soci ja té 3 préstecs.
post *prestar*: Una còpia qualsevol de la biblioteca del soci que estigui disponible passa a estar en préstec amb deixat=data d'avui (podeu suposar l'existència de l'operació avui():Date que retorna la data d'avui i que pot ser invocada des d'on la necessiteu). Es forma l'associació Té entre el soci i el préstec. S'incrementa en 1 l'atribut ncòpiesenpréstec de la bilbioteca del soci.

Es demana:
a) Diagrama de seqüència de l'operació de sistema *obt_dades_préstec* i *fer_préstec*, suposant que la navegabilitat de l'associació Pot_oferir és doble.
b) Diagrama de classes resultant (només paquet domain model). No cal posar les operacions. Indiqueu clarament les navegabilitats.

Exercici 17 (Patrons)

Una administració de finques ens ha demanat que dissenyem un sistema per gestionar les vendes actuals de pisos. Dels pisos s'enregistra la seva adreça, que els identifica, l'any de construcció i el venedor. Els pisos poden estar en venda o venuts. Dels pisos en venda sabem el preu de venda i dels pisos venuts sabem el comprador i la data de la compra. El sistema enregistra les persones que són propietaris o compradors dels pisos. D'aquestes persones en sabem el dni que les identifica i el seu nom. A continuació disposeu de l'esquema conceptual del sistema.



R.I. Textuals:

- Claus: (Pis, adreça); (Persona, dni);
- L'atribut preuVenda ha de ser superior a 0.
- L'atribut anyConstrucció ha de ser igual o superior a 1950.
- L'any de la data de venda d'un pis ha de ser posterior a l'any de construcció
- El comprador i venedor d'un pis no pot ser el mateix.

Volem dissenyar dues operacions de la capa de domini:
context CapaDomini:: vendrePis (adreça: String, dniComprador: String, dataVenda: Date)
pre *dataVendaOk:* la data de la venda és correcte.
exc *pisNoExisteix:* el pis amb adreça *adreça* no existeix.
exc *pisVenut:* el pis amb adreça *adreça* ja està venut.
exc *compradorNoExisteix:* el comprador amb dni *dniComprador* no existeix.
exc *jaÉsPropietari:* el comprador amb dni *dniComprador* és el venedor del pis.
post *passaAVenut: el pis amb* adreça *adreça* passa a estar venut i se li assigna la dataVenda i el comprador.

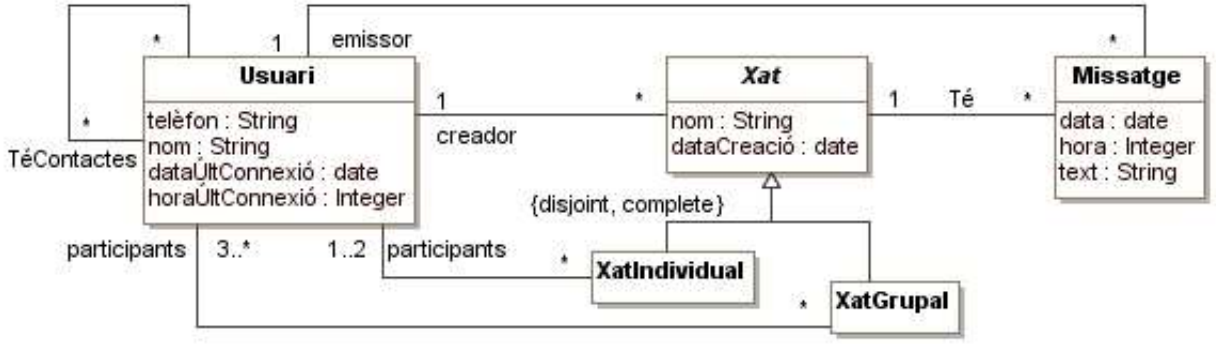
context CapaDomini:: pisEnVenda (adreça: String, anyConst: Integer, dniVenedor: String, preuVenda: Integer)
exc *pisJaExisteix:* el pis amb adreça *adreça* ja existeix.
exc *anyConstrucciónoOk:* l'any de construcció és inferior a 1950.
exc *venedorNoExisteix:* el venedor amb dni *dniVenedor* no existeix.
exc *preuVendaNoOk:* el *preuVenda* és 0 o negatiu.
post *altaPisEnVenda: es crea el pis en venda i se li assigna l'adreça, l'any de construcció el preu de venda i el venedor.*

Es demana:

a) Diagrama de seqüència de l'operació de les dues operacions, suposant que les navegabilitats són de Pis a Persona.
b) Diagrama de classes resultant (només paquet domain model).

Exercici 18 (Patrons Factoria, Singleton i Adaptador)

Volem dissenyar un sistema software de missatgeria. Aquest software tindrà usuaris i els seus contactes. Cada usuari podrà crear un nombre indeterminat de xats. Els xats podran ser xats individuals i grupals. Els xats individuals tindran com a molt dos participants, el creador i un altre participant i els grupals tindran un mínim de 3 participants. Els xats tindran els missatges que envien els seus emissors (que han de ser participants del xat). A continuació disposeu d'un fragment de l'esquema conceptual del sistema a dissenyar:
Esquema conceptual de l'especificació:



R.I. Textuals:
- Claus: (Usuari, telèfon); (Xat, Usuari::telèfon + nom); (Missatge, Xat::nom +Usuari::telèfon + data + hora);
- El creador d'un xat ha de ser un dels seus participants.
- L'emissor d'un missatge d'un xat ha de ser un dels participants del xat.
- La data d'un missatge ha de ser posterior a la data de la creació del xat.
- Els participants dels xats d'un usuari han de ser contactes de l'usuari del xat i ell mateix.
- Altres restriccions no rellevants pel problema

Volem dissenyar l'operació de la capa de domini *EliminarParticipantXat* per eliminar un dels participants del xat. A continuació disposeu del contracte de l'operació:

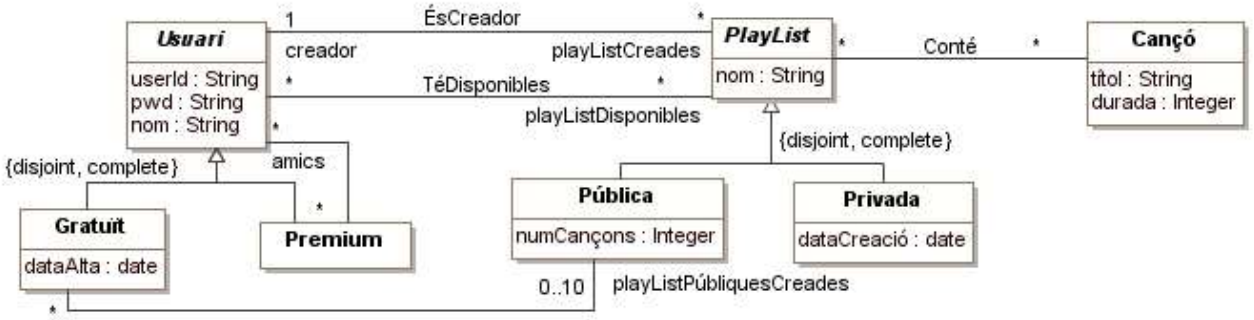
context CapaDomini :: EliminarParticipantXat (nomXat: String, usuariCreador: String, usuariParticipant: String)
exc *xatNoExisteix*: El xat identificat per *nomXat* i *usuariCreador* no existeix.
exc *noParticipa*: L'usuari *usuariParticipant* no és un participant del xat.
exc *usuariAmbMissatges*: L'*usuariParticipant* és emissor de missatges al xat.
exc *usuariCreador*: L'*usuariParticipant* és el creador del xat.
exc *usuariAmbPocsXats*: L'*usuariParticipant* participa en menys de 3 xats.
exc *xatAmbParticipantsMínims*: El xat té el nombre de participants mínim (3 si és grupal i 1 si és individual).
post *eliminaParticipant*: S'elimina el participant del xat.

Suposant que totes les navegabilitats són dobles, es demana:
- Diagrama de seqüència de l'operació EliminarParticipantXat de la capa de domini. Indiqueu clarament en el diagrama de seqüència els singleton, les interfaces i les operacions que són abstractes. Suposeu que totes les navegabilitats són dobles excepte les de les associacions entre els diferents tipus de xat i els seus participants. Aquestes navegabilitats van del tipus de xat cap a l'usuari.

Exercici 19 (Patrons Factoria, Singleton i Adaptador)

Volem dissenyar un sistema software per gestionar la música d'un conjunt d'usuaris. Aquest software permetrà als seus usuaris crear playlists que podran ser públiques o privades. Les playlists tindran el seu usuari creador, un nom, i les seves cançons. Els usuaris podran ser gratuïts o premium. Els usuaris premium podran enregistrar altres usuaris amics. Cada usuari serà creador de les seves pròpies playlists però podrà tenir disponibles, a més, altres playlists dels seus amics. A continuació disposeu d'un fragment de l'esquema conceptual del sistema a dissenyar:

Esquema conceptual de l'especificació:



R.I. Textuals:
- Claus: (Usuari, userId); (PlayList, Usuari::userId del creador+ nom); (Cançó, títol);
- Un usuari premium no pot tenir com amic a sí mateix.
- Un usuari gratuït no pot tenir disponibles més de 20 playlists.
- Un usuari gratuït no pot tenir més de 30 cançons en les playlists que té disponibles.
- Les playlists que crea un usuari són un subconjunt de les playlists que té disponibles.
- Les playlists públiques creades per un usuari gratuït són un subconjunt de les playlists que ha creat.
- L'atribut numCançons d'una playlist pública és igual al nombre de cançons que té assignada la playlist.
- Altres restriccions no rellevants pel problema

Volem dissenyar dues operacions de la capa de domini. L'operació *assignarPlayList* de la classe *Usuari* que permet assignar una playlist com a disponible a un usuari i *crearICompartirPublicPlayList* per crear una playlist pública d'un usuari i assignar-la com a disponible als seus amics. A continuació disposeu del contracte de les dues operacions:

context Usuari :: assignarPlayList (pl: PlayList)
exc *playListAssignada*: La playlist *pl* ja la té disponible l'usuari *self*.
exc *gratuïtAmbTotesPL*: L'usuari *self* és gratuït i té disponibles 20 playlists.
exc *gratuïtiMoltesCançons*: L'usuari *self* és gratuït i el nombre de *cançons* del *pl* és més gran que 30.
post *assignacióPLDisponible*: S'assigna a l'usuari *self* la playlist *pl* com a playlist disponible.

context CapaDomini :: crearICompartirPublicPlayList (userIdCreador: String, nomPL: String, cançons: Set(títol:String))
pre *cançonsExisteixen*: Les cançons del conjunt *cançons* existeixen i el nombre de cançons és més gran que 0.
exc *userNoExisteix*: L'usuari identificat per *userIdCreador* no existeix.
exc *playListExistent*: La playlist identificada per *userIdCreador* i *nomPL* ja existeix.
exc *gratuïtAmbTotesPL*: L'usuari identificat per *userIdCreador* és gratuït i té 20 playlists disponibles.
exc *gratuïtiMoltesCançons*: L'usuari identificat per *userIdCreador* és gratuït i el nombre de *cançons* és més gran que 30.

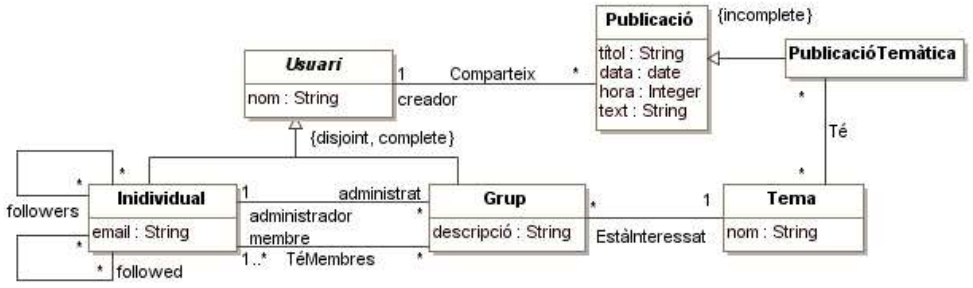
exc *gratuïtiMoltesPúbliquesCreades*: L'usuari identificat per *userIdCreador* és gratuït i ja té 10 playlists públiques creades.
post *creacióPL*: Es crea la playlist pública amb el nom, i s'assignen les cançons i l'usuari creador indicat als paràmetres.
post *assignacióPLDisponible*: S'assigna la playlist com a playlist disponible per a l'usuari creador.
post *assignacióPLPúblicaAGratuït*: Si l'usuari creador és gratuït llavors s'assigna la playlist creada a les *playListPúbliquesCreades* per l'usuari.
post *assignacióPLAmics*: Si l'usuari creador és premium, s'assigna la playlist creada a la llista de playlist disponibles de tots els seus usuaris amics que siguin premium i, si la playlist pública creada té 30 o menys cançons, s'assigna la playlist també a tots els amics gratuïts que no tinguin 20 playlists disponibles.

Suposant que les navegabilitats inicials de l'associació TéDisponibles és doble i la navegabilitat d'Usuari (rol creador) va cap a PlayList, la de Gratuït va cap a Pública i la de l'associació Conté va de PlayList a Cançó (recordeu que com a conseqüència del vostre disseny, si és necessari, podeu afegir noves navegabilitats), es demana:

a) Diagrama de seqüència de l'operació assignarPlayList de la classe Usuari. Indiqueu clarament en el diagrama de seqüència els singleton, les interfaces i les operacions que són abstractes.
b) Diagrama de seqüència de l'operació crearICompartirPublicPlayList de la capa de domini. Indiqueu clarament en el diagrama de seqüència els singleton, les interfaces i les operacions que són abstractes. Es valorarà especialment la reutilització de les operacions que dissenyeu. En concret, aquest sistema haurà de permetre que un usuari premium pugui crear una playlist pública sense compartir-la amb els seus amics.

Exercici 20 (Patrons Factoria, Singleton i Adaptador)

Volem dissenyar una xarxa social per compartir publicacions. La xarxa disposarà de dos tipus d'usuaris que s'identificaran amb el nom. Els usuaris poden ser individuals o de grup. Els usuaris de grup disposen d'un nom, d'una descripció, d'un usuari individual que l'administra, d'usuaris individuals que són membres del grup i del tema en el que estan interessats. Els usuaris individuals disposen d'un nom, d'un email i tenen seguidors i seguits. Tots els usuaris poden compartir publicacions. Les publicacions que tenen un títol, un text, una data i hora de publicació poden ser temàtiques o no. Les publicacions temàtiques tenen un conjunt de temes assignats. A continuació disposeu de l'esquema conceptual del sistema a dissenyar:



R.I. Textuals:
- Claus: (Usuari, nom); (Publicació, títol); (Tema, nom);
- Un usuari de grup només pot compartir publicacions temàtiques que tinguin almenys un tema que coincideixi amb el tema en el que l'usuari de grup està interessat.
- Un usuari individual no es pot seguir a si mateix i no pot ser seguit per si mateix.

- L'usuari administrador d'un usuari de grup és un dels usuaris individuals que és membre.
- Altres restriccions no rellevants pel problema

Volem dissenyar l'operació de la capa de domini *compartirPublicacióTemàtica*. Aquesta operació crea una publicació temàtica i obté els interessats en les publicacions de l'usuari creador. A continuació disposeu del contracte de l'operació:

context CapaDomini:: compartirPublicacióTemàtica (nomUsuariCreador: String, títol: String, text: String, temes: Set(nom:String)): TupleType(nbPub: Integer, interessatsPubs: Set(nomUsuari: String))
pre *hiHaTemes:* el conjunt *temes* té algun tema.
exc *temaNoExisteix:* algun dels temes del conjunt *temes* no existeixen.
exc *usuariNoExisteix:* l'usuari amb nom *nomUsuariCreador* no existeix.
exc *publicacióTemàticaExisteix:* la publicació temàtica amb títol *títol* existeix.
exc *usuariNoPotCompartir:* l'usuari creador és individual i no té followers o l'usuari és de grup i té només un membre.
exc *publicacióDeGrupSenseTema:* l'usuari creador és de grup i el conjunt de temes no té el tema en el que està interessat el grup.
post *creacióPublicacióTemàtica:* es dóna d'alta la publicació temàtica i se li assignen els temes, el creador, el títol, el text, la data i l'hora actual (la data i l'hora actual es poden obtenir invocant a les operacions *getData():date* i *getHora(): Integer* des d'on les necessiteu).
post *result* = es retorna el nombre de publicacions de l'usuari creador i el nom dels interessats en les publicacions que comparteix l'usuari creador. Aquests interessats seran:
- Si l'usuari creador és de grup, els usuaris que són membres del grup.
- Si l'usuari creador és individual, els seus followers i els seus companys en tots els usuaris de grup dels que és membre.

Suposant que les navegabilitats inicials de l'associació Comparteix és doble i la navegabilitat de l'associació Té va de PublicacióTemàtica a Tema (recordeu que com a conseqüència del vostre disseny, si és necessari, podeu afegir noves navegabilitats), es demana:

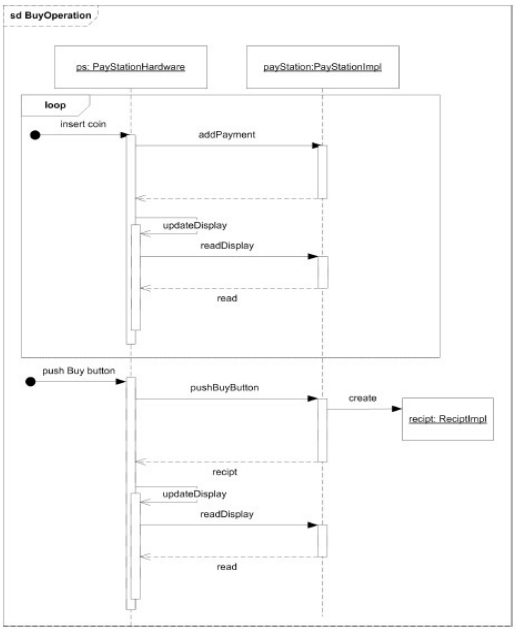
- Diagrama de seqüència de l'operació compartirPublicacióTemàtica. Indiqueu clarament en el diagrama de seqüència els singleton, les interfaces i les operacions que són abstractes. Heu de considerar que:
  - En un altre cas d'ús del sistema és necessari crear publicacions temàtiques (sense obtenir els interessats en les publicacions de l'usuari creador) i, per tant, cal que aquesta operació constructora sigui altament reusable.

Exercici 21 (Patrons)

Volem dissenyar un sistema software pels parquímetres de la ciutat de Nova York. Aquest software implementarà inicialment 3 històries d'usuari:

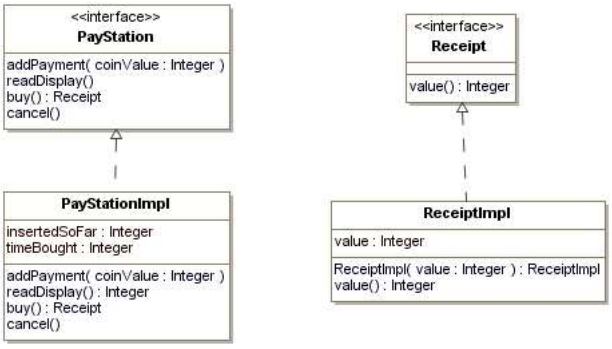
- Comprar un tiquet de parquing. Aquesta història s'inicia quan un conductor introdueix diferents monedes vàlides (5, 10 i 25 centaus de dolar) al parquímetre. Per cada 5 centaus de dòlar, el parquímetre proporciona 2 minuts de temps. Quan el conductor està satisfet amb el temps comprat (i visualitzat en la pantalla del parquímetre), prem el botó *buy* per fer la compra. El conductor rep un tiquet amb els minuts comprats i el display del parquímetre es torna a posar a 0. A continuació

disposeu del diagrama de seqüència d'aquesta història d'usuari que mostra la interacció entre el hardware (parquímetre) i el software que volem dissenyar.



- Cancel.lar la compra d'un tiquet de parquing. Si el conductor està introduint monedes i decideix cancel.lar la compra, prem el botó *cancel.lar* i el parquímetre li retorna les monedes introduides i el display es torna a posar a 0.
- Rebutjar una moneda no permesa. Si el conductor introdueix una moneda no permesa, el parquímetre retorna la moneda i el display no s'actualitza.

A continuació disposeu d'un fragment del diagrama de classes del sistema a dissenyar:



Es demana:

1. Diagrama de seqüencia de les operacions de la classe PayStationImpl().
2. Una vegada dissenyades les operacions, ens arriba un nou requisit que hem d'encabir en el nostre sistema. Hem rebut la petició d'una altra ciutat (per exemple Boston) per comprar el software del PayStation però amb un nou requisit sobre com calcular les tarifes. En concret, Boston vol aplicar una tarifa progressiva d'acord amb el següent esquema (és possible que més endavant Boston vulgui mantenir la tarifa progressiva de dilluns a divendres i la tarifa linial de Nova York el cap de setmana):

- Primera hora: $1,5 (5 cents gives 2 minutes)
- Segona hora: $2 (5 cents gives 1,5 minutes)

- Tercera i successives hores: $3 (5 cents gives 1 minute)

Volem mantenir i donar suport al software del PayStation que s'estarà executant en dos ciutats diferents amb un comportament molt similar. És possible que en el futur altres ciutats estiguin interessades en comprar el nostre software amb un càlcul de tarifes que pot ser diferent. El problema que es planteja és el següent: Com podem desenvolupar i mantenir dos o més variants d'aquest software amb el mínim cost i els mínims defectes? Descriviu 2 propostes diferents (diagrama de classes i els diagrames de seqüència de les operacions que es vegin modificades respecte a les proporcionades a l'apartat anterior) utilitzant els patrons estudiats a classe per resoldre el problema anterior i discutiu els avantatges i inconvenients de cada proposta.