ARQUITECTURA DEL SOFTWARE

Unit 2.2: Object Oriented Architecture

Exercicis Resolts

Exercici 1

Un dels principis de disseny que han de complir les arquitectures orientades a objecte és el baix acoblament. Fes una taula on es descrigui el tipus d'acoblament (mireu *Ingeniería del software. Un enfoque práctico.* R.G. Pressman. McGraw Hill, 2010), en quin/s diagrames es pot detectar el tipus d'acoblament i una possible manera d'evitar-ho.

Solució

Tipus acoblament	Descripció	Problema	Es detecta a	Com evitar-ho
Acoblament de contingut	Un component modifica dades internes d'altres components	Canvis a les dades d'un component poden afectar a altres components	DC i DS	Dades privades i modificació d'aquestes via operacions
Acoblament comú	Uns components fan ús de var. globals	Propagació d'errors quan es canvien v. globals	DS	No definir variables globals (ús Singleton)
Acoblament del control	Una operació *A* invoca a *B* i li passa una var. de control que dirigeix la lògica de *B*	Un canvi de la var. de control pot provocar un error en *B*.	DS	Evitar, sempre que es pugui, les variables de control (ús polimorfisme)
Acoblament de motllo	Es declara una classe *B* com un tipus d'argument d'una operació d'*A*.	La modificació es torna més complexa ja que *A* coneix a *B*.	DC i/o DS	Evitar aquest tipus de declaracions
Acoblament de dades	Es passen cadenes llargues d'arguments de dades	La complexitat creix i es dificulta fer proves i manteniment	DC i/o DS	Evitar aquest arguments

Tipus acoblament	Descripció	Problema	Es detecta a	Com evitar-ho
Acoblament de crida	Un operació invoca a una altra	Incrementa la connectivitat del sistema	DS	És comú i a vegades necessari
Acoblament de tipus d'ús	Un component *A* usa un tipus de dades definit a un component *B* (tipus d'atributs)	Un canvi a *B* pot implicar un canvi a *A*	DC	És comú i a vegades necessari
Acoblament d'inclusió o importació	Un component *A* importa o inclou un paquet o el contingut d'un component *B*	Un canvi a *B* pot implicar un canvi a *A*	DC	Comú (respectar els principis de disseny de paquets, veure *unit 2.1*)
Acoblament extern	Un component es comunica amb components d'infraestructura (SO, BD)	Dependència dels components d'infraestructura	DS	Necessari però s'ha de limitar a un nombre petit de classes o components del sistema.

Exercici 2

Considera el diagrama de classes de la transparència 13 del tema 2.2. Explica, de forma textual (no cal fer el contracte), com es crea una venda (*Sale*).

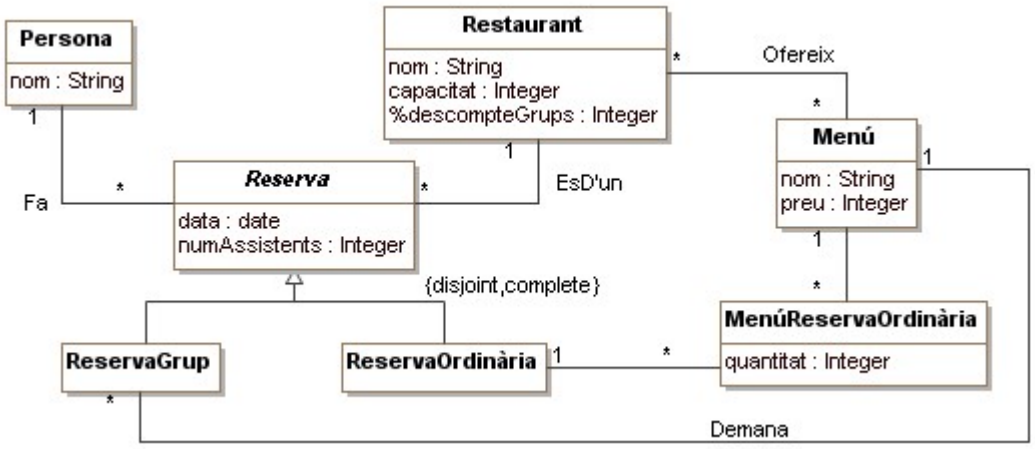
Solució

Al crear una *Sale* cal:

- Inicialitzar els seus atributs
- Donar d'alta l'estratègia de preus. Per fer això cal donar d'alta o obtenir la instància concreta de *PercentDiscountPricingStrategy* o *AbsoluteDiscountOverThresholdPricingStrategy* o etc… i crear una instància de l'associació entre *Sale* i aquesta instància.

Exercici 6

Una cadena de restaurants que ofereixen menús per sopars ens ha demanat que li dissenyem una part d'un sistema software per gestionar les seves reserves. Els clients de la cadena fan reserves per un restaurant concret en una data determinada. Les reserves poden ser de dos tipus: les reserves de grup són reserves per a un conjunt de persones que demanen el mateix menú i ho fan en el moment de fer la reserva, i les reserves ordinàries són reserves per a una o més persones que demanen el menú que vulguin en el moment de sopar. Els restaurants ofereixen un descompte (%descompteGrups) en el preu del menú per a les reserves de grup. A continuació disposeu de l'esquema conceptual i dels contractes de les operacions a dissenyar:



R.I. Textuals:
- Claus: (Persona, nom); (Restaurant, nom); (Menú, nom); (Reserva, nomPersona+data)
- No poden haver-hi dos menús de reserva ordinària pel mateix menú i reserva ordinària.
- Els menús de les reserves 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 d'aquell restaurant per una data determinada.
- Tots els atributs de tipus integer han de ser positius
- El numAssistents d'una reserva de grup ha de ser més gran que 5
- En una reserva ordinària el numAssistents ha de coincidir amb la quantitat de menús demanats
- Altres restriccions no rellevants pel problema

context CapaDomini::preuReserva(nomP: String, data:date): Float
pre *té-reserva*: la persona té una reserva a la data indicada al paràmetre.

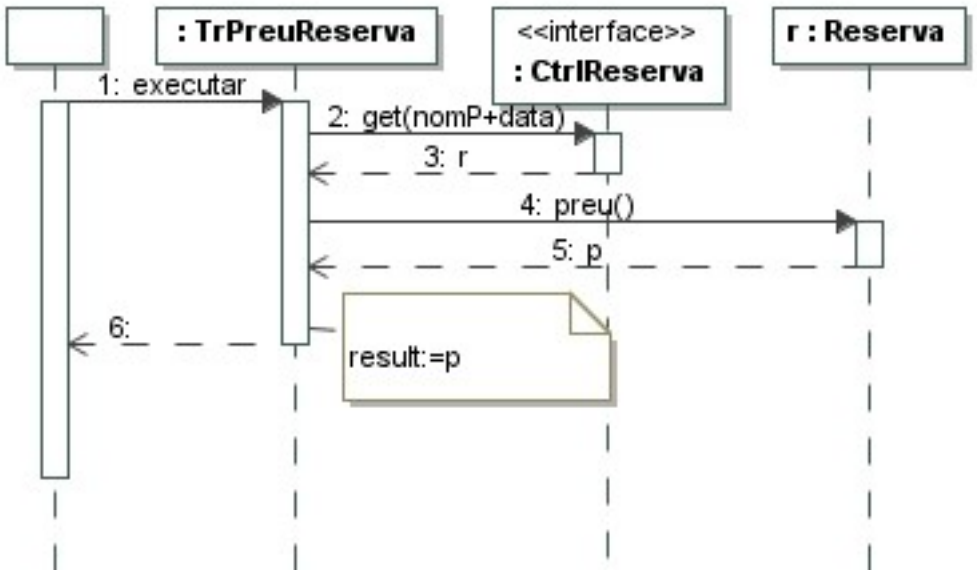
post result= preu de la reserva associada a la persona *self* en la data indicada al paràmetre. El preu d'una reserva es calcula de la següent forma:
1) si la reserva és de grup aleshores preu= numAssistents*preu menú demanat - %descompteGrups del restaurant/100*(numAssistents*preu menú demanat)
2) si la reserva és ordinària aleshores preu= ∑ quantitat*preu menú per a cada menú demanat.

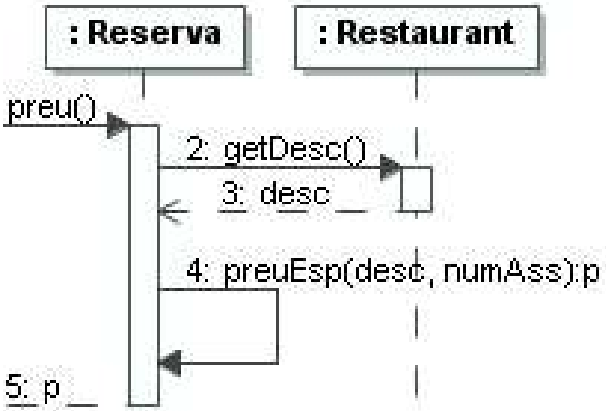
context CapaDomini::novaReservaGrup(nomP: String, nomR: String, dr: date, nAss: Integer, nomMenú: String)
pre *persona-existeix*: la persona *nomP* existeix.
pre *restaurant-existeix*: el restaurant *nomR* existeix.
pre *numAssistents-ok*: el *nAss* és més gran que 5.
pre *restaurant-amb-capacitat*: el restaurant *r* té capacitat per acollir la reserva.
exc *ja-existeix-reserva*: la persona ja ha fet una reserva per la data *dr*.
exc *menú-no servit*: el menú *nomMenú* no es ofert pel restaurant *nomR*.
post es crea una reserva de grup i es formen totes les associacions amb la persona, el restaurant i el menú.

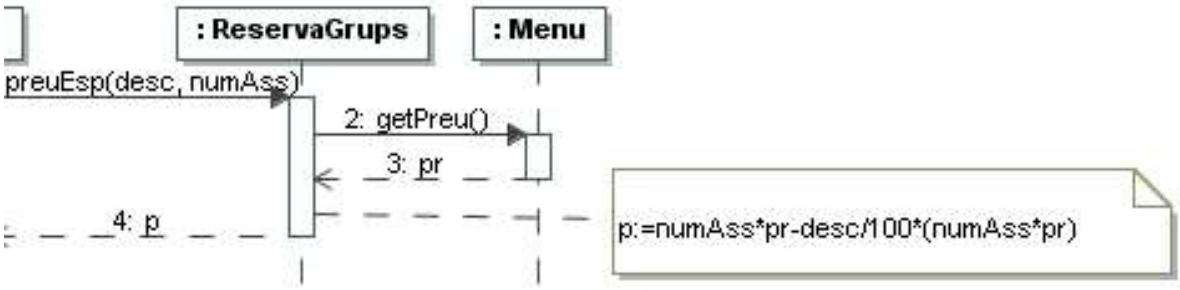
Es demana:
a) Diagrama de seqüència de l'operació *preuReserva* i *novaReservaGrup*.
b) Diagrama de classes de la capa de domini.

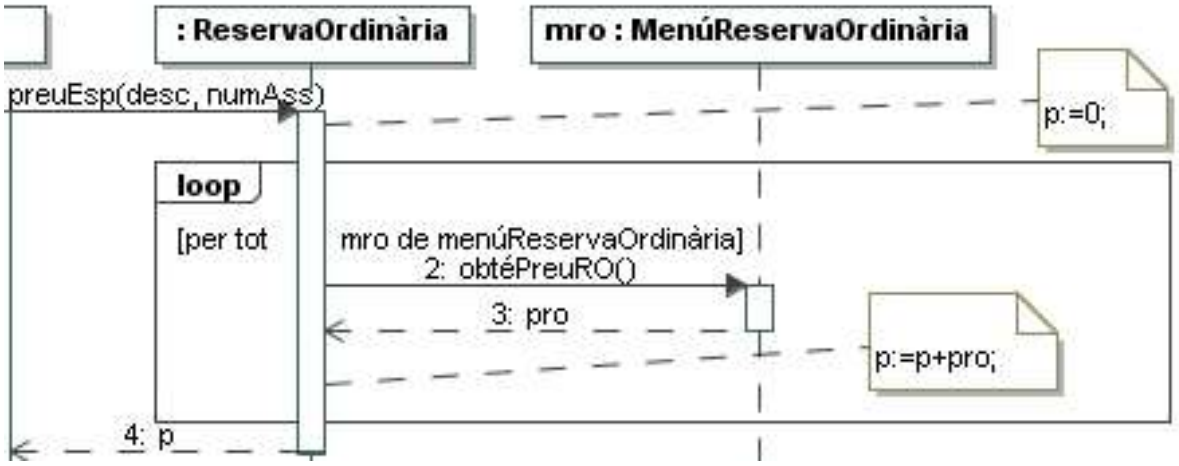
Solució

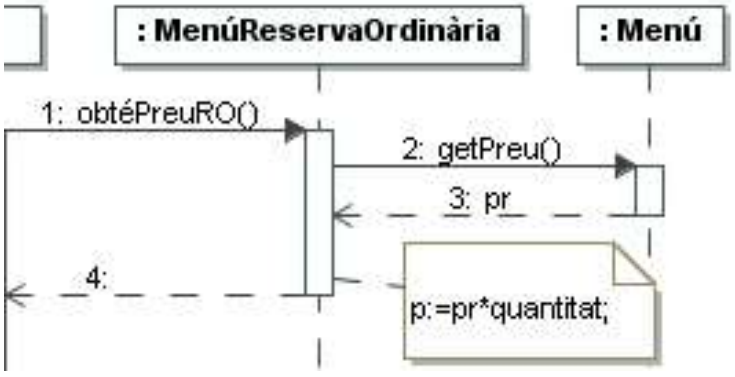
a) preuReserva



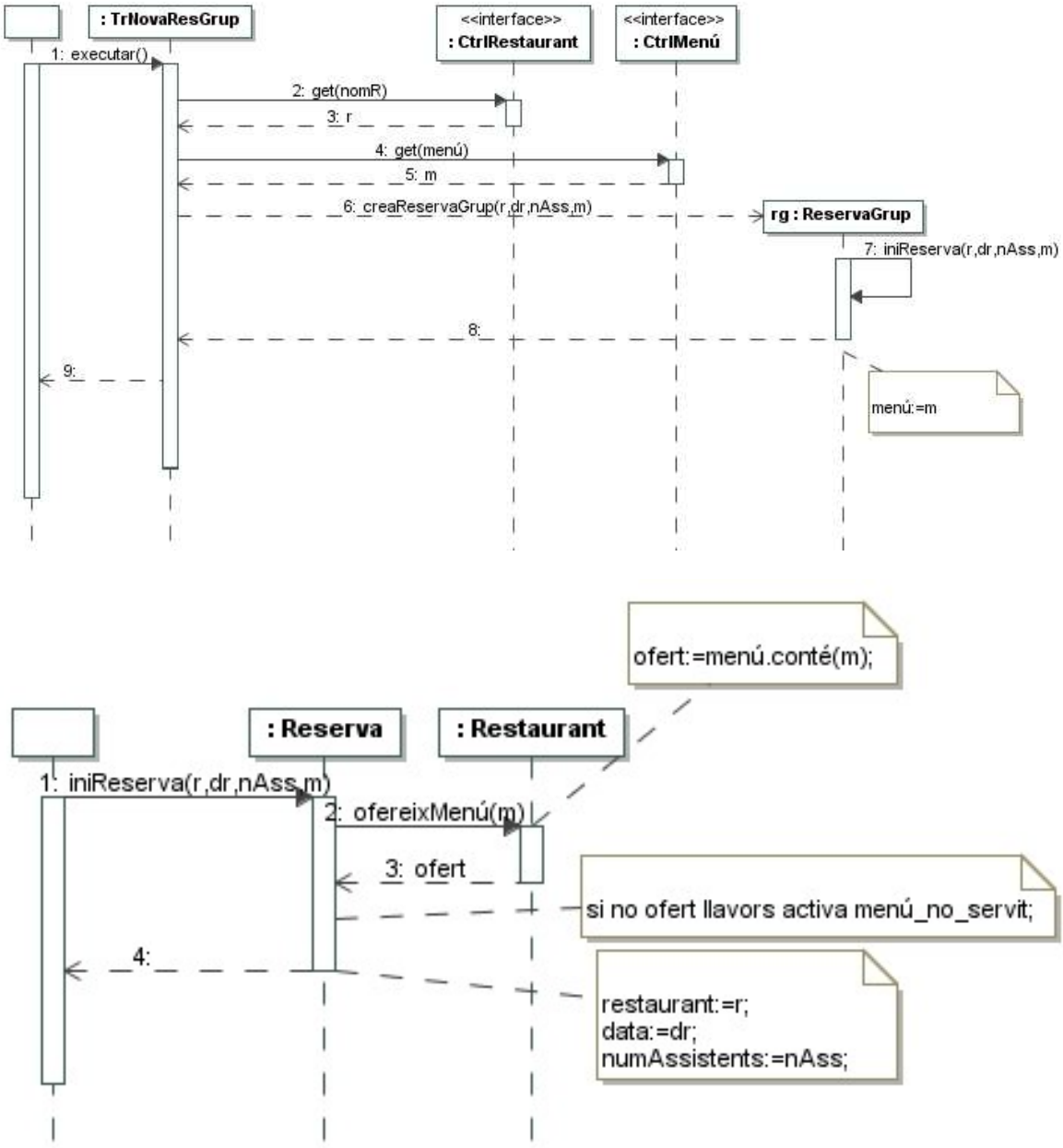

: Reserva
: Restaurant
preu()
2: getDesc()
3: desc
4: preuEsp(desc, numAss):p
5: p


: ReservaGrups
: Menu
preuEsp(desc, numAss)
2: getPreu()
3: pr
4: p
p:=numAss*pr-desc/100*(numAss*pr)

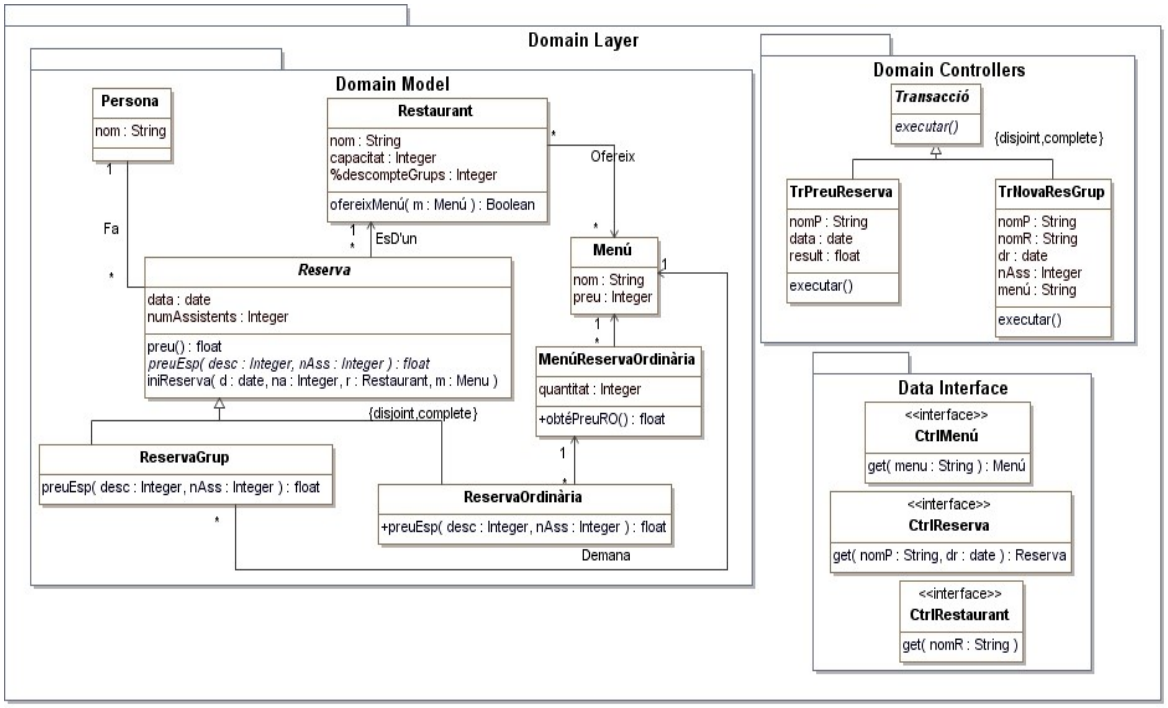

: ReservaOrdinària
mro : MenúReservaOrdinària
preuEsp(desc, numAss)
p:=0;
loop
[per tot
mro de menúReservaOrdinària]
2: obtéPreuRO()
3: pro
p:=p+pro;
4: p


: MenúReservaOrdinària
: Menú
1: obtéPreuRO()
2: getPreu()
3: pr
4:
p:=pr*quantitat;

a) novaReservaGrup

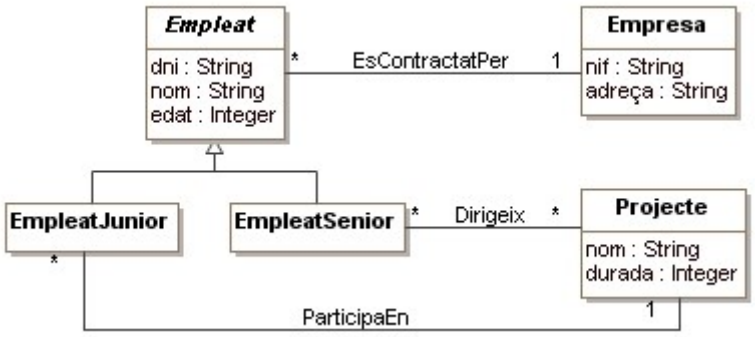


b) Diagrama de classes de la capa de domini



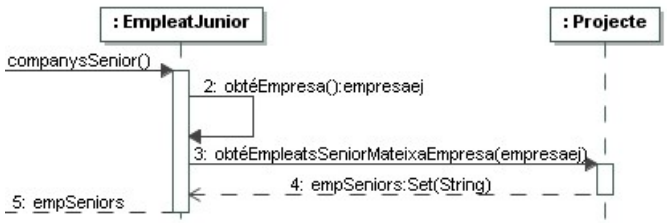
Exercici 7

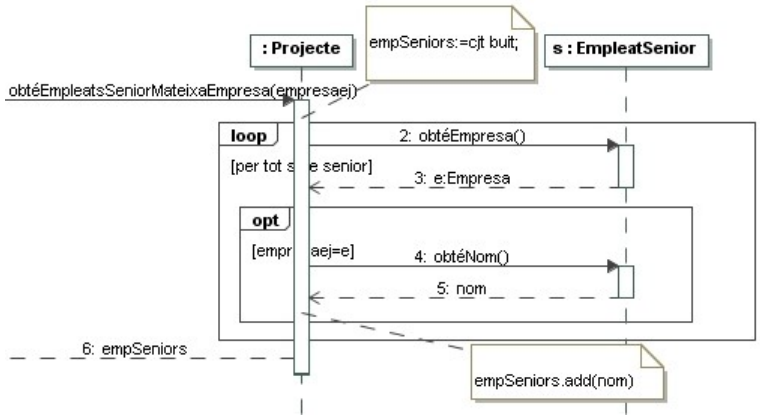
(Competència transversal) Donat l'esquema conceptual i els diagrames de seqüència següents:

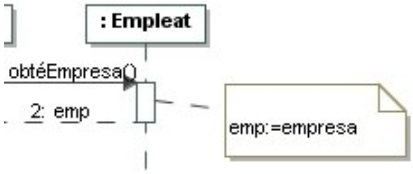


R.Integritat Textuals:
- Claus classes no associatives: (Empresa, nif); (Empleat, dni); (Projecte, nom)

context Junior::companysSenior(): Set(String)
post resultat= nom dels empleats seniors que dirigeixen el projecte on *self* participa i que estan contractats per la mateixa empresa.





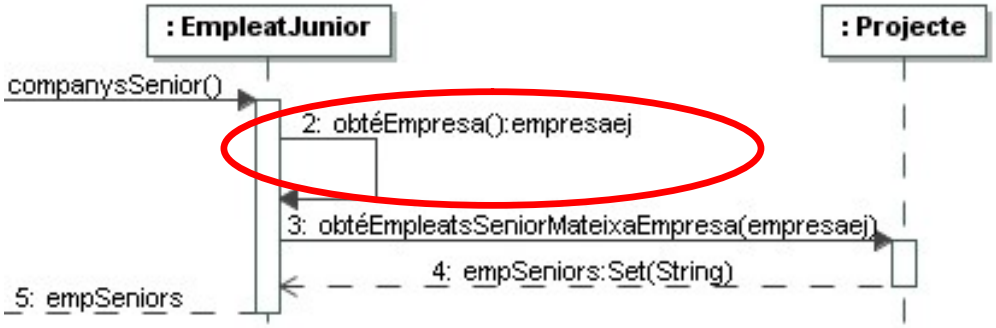


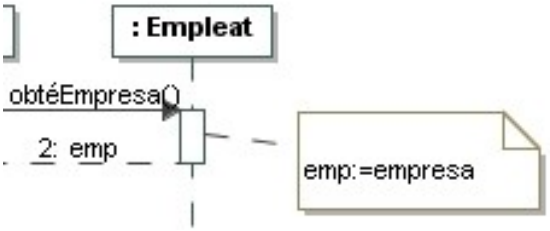
Es demana:

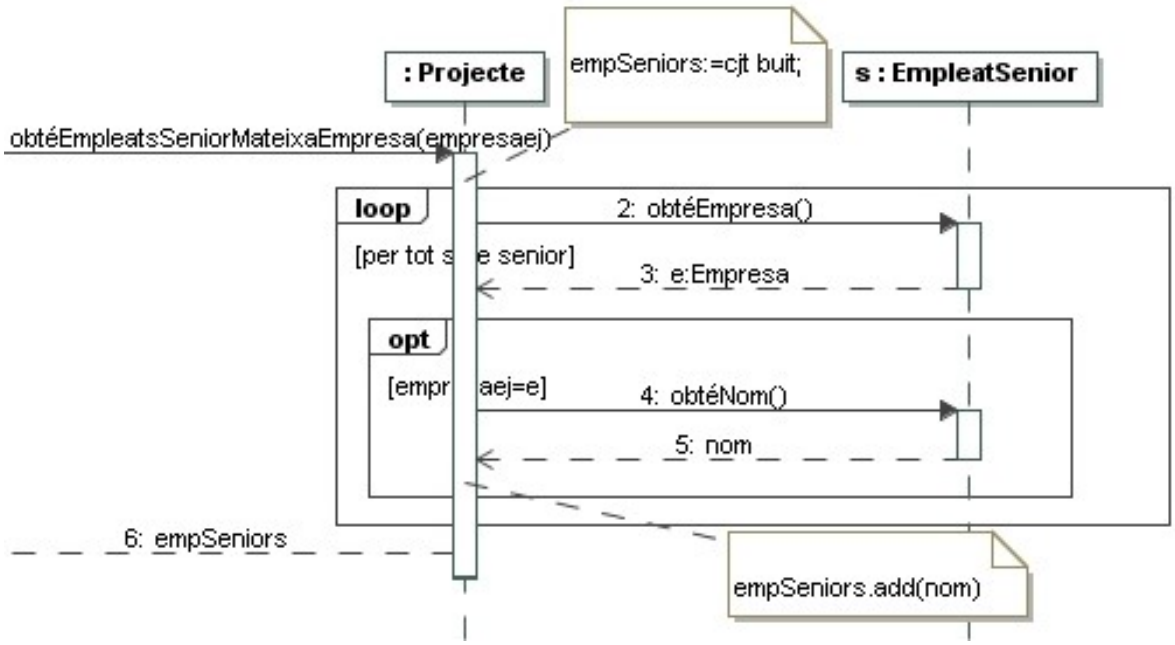
a) Raona i justifica si el diagrama de seqüència anterior aplica el principi de disseny de *baix acoblament*.
b) Proposa una solució alternativa a l'anterior que millori l'acoblament.

Solució

a)





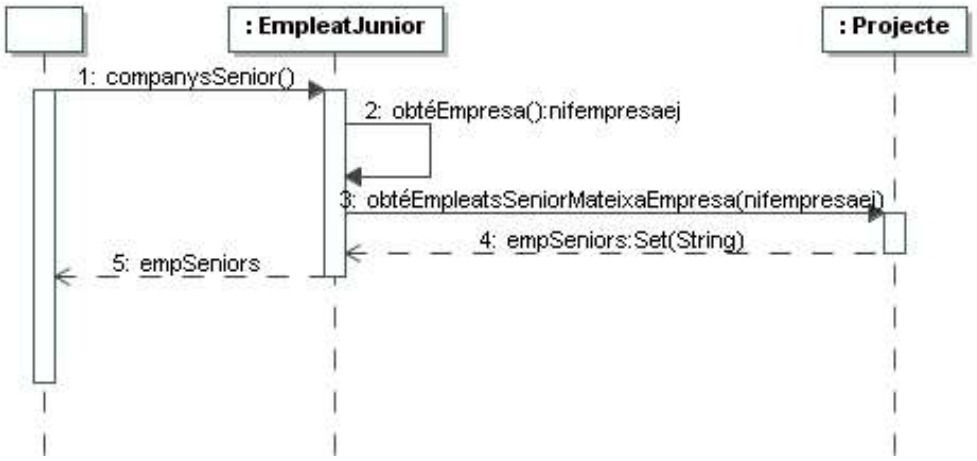


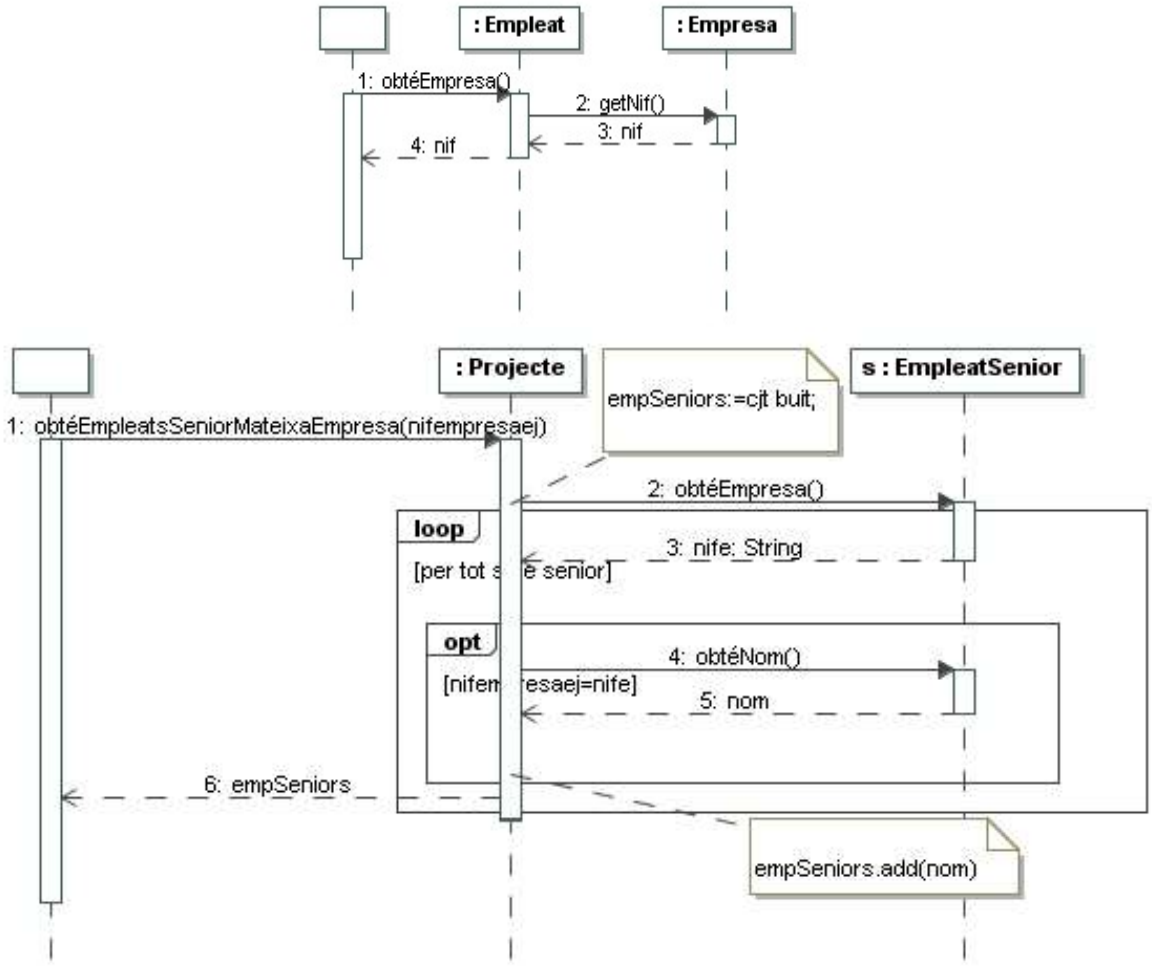
El disseny de l'operació obtéEmpresa genera un alt acoblament. Al retornar l'objecte empresa provoquem acoblament entre:

- EmpleatJunior i Empresa
- EmpleatSenior i Empresa
- Projecte i Empresa

Aquests acoblaments són introduïts com a conseqüència del disseny de l'operació. Cal redissenyar l'operació per tal de minimitzar aquests acoblaments.

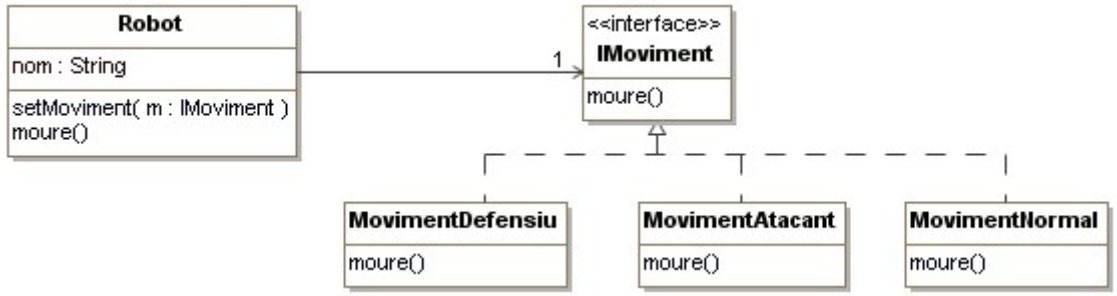
b)





Exercici 8

Un laboratori de disseny de robots ens ha demanat que li dissenyem una part d'un sistema software per simular els diferents moviments que fan els robots que dissenyen. Els robots tenen un nom i realitzen moviments en funció d'una estratègia de moviment (que pot ser defensiva, atacant o normal). A continuació disposeu del diagrama de classes del paquet *domain model* de la capa de domini i dels contractes d'un conjunt d'operacions de la capa de domini.



R.I. Textuals:
- Claus: (Robot, nom)

context CapaDomini::crearRobotAmbMovimentsDefensius (nom:String)
exc *robot-existeix*: el robot amb nom *nom* existeix.
post *crea-robot***:** es crea el robot amb nom *nom* amb moviments defensius.

context CapaDomini::canviarAMovimentAtacant (nom:String)
exc *robot-no-existeix*: el robot amb nom *nom* no existeix.
post *modifica-moviment***:** es crea un el moviment atacant i s'assigna al robot.

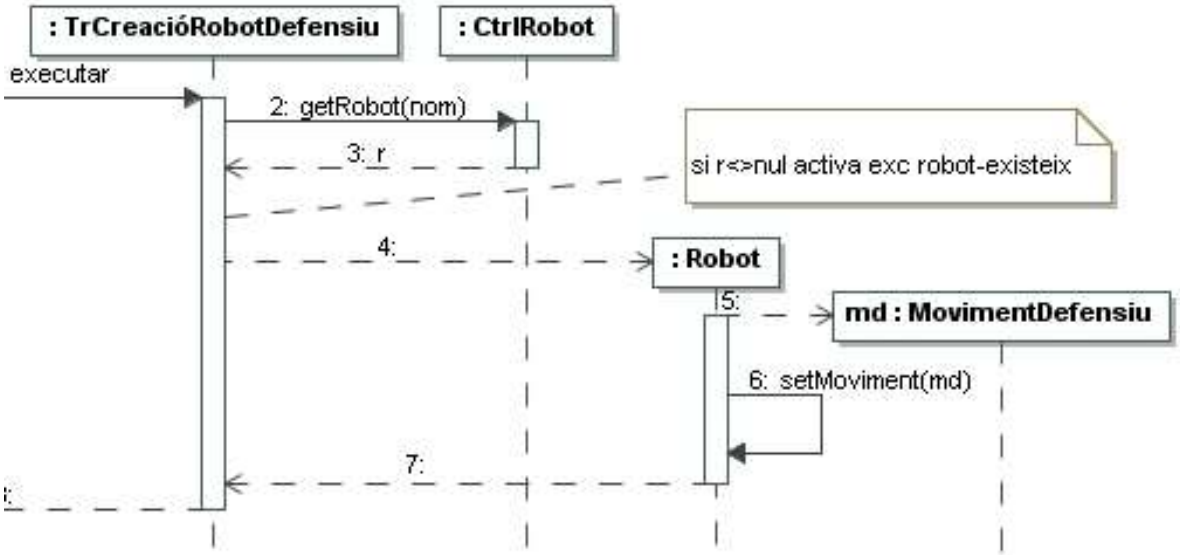
context CapaDomini::moureRobot(nom:String)
exc *robot-no-existeix*: el robot amb nom *nom* no existeix.
post es mou el robot segons el moviment que tingui assignat.

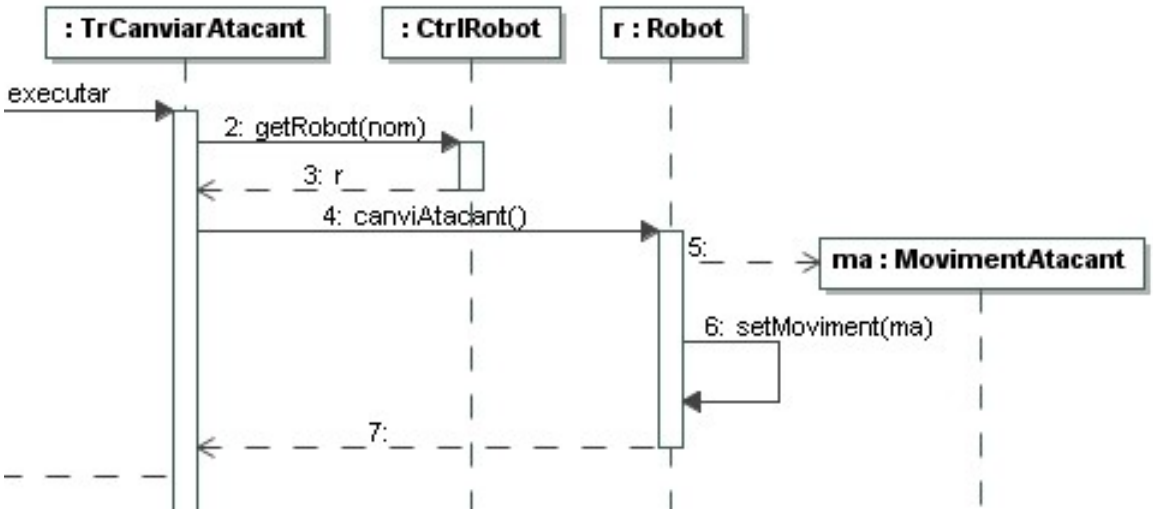
Es demana:

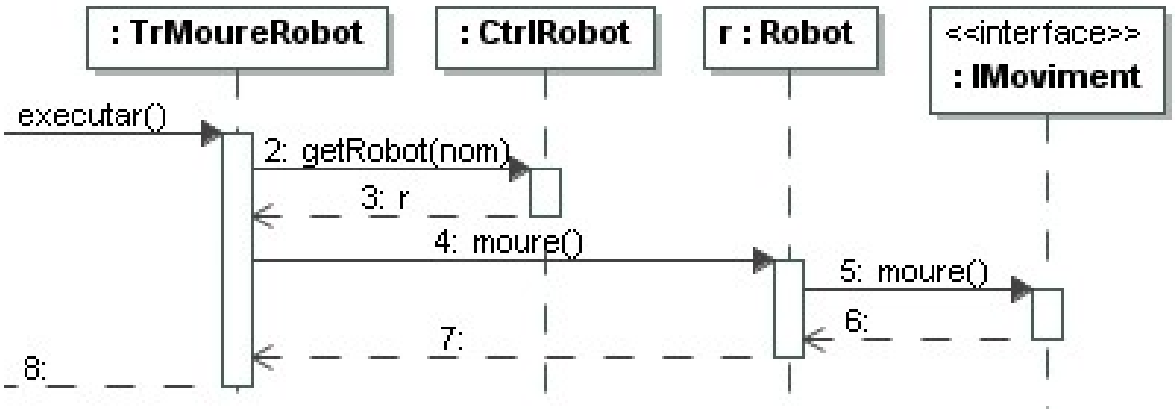
a) Diagrama de seqüència de les operacions anteriors.

Solució

a)

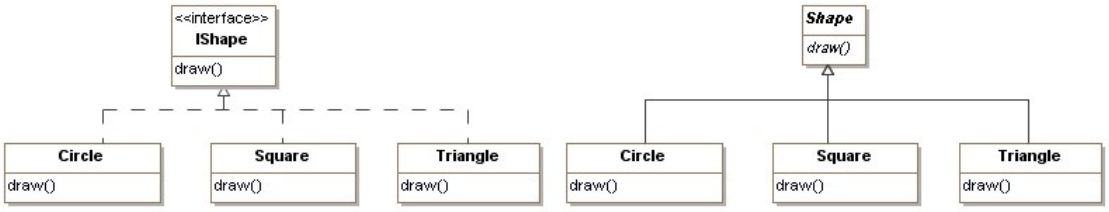






Exercici 9

(Competència transversal) Ens han demanat que dissenyem un editor gràfic que ha de tenir funcionalitats per dibuixar figures senzilles. A continuació disposeu de dues alternatives d'un fragment del diagrama de classes del paquet *domain model* de la capa de domini.



Es demana:

a) Raona i justifica quina de les alternatives et sembla més adient en els següents supòsits:
   a. Tal i com es mostra a la figura, la implementació de l'operació *draw* és diferent per a cada figura.
   b. L'operació *draw* es modifica i passa a tenir una part igual per a totes les figures i una altra part diferent per a cada una d'elles.
   c. Està previst en un futur afegir l'atribut privat *center* (que indica el punt central de la figura) a *Shape*.
   d. Està previst en un futur afegir noves operacions públiques a *Circle, Square i Triangle* que heretin d'una classe A.

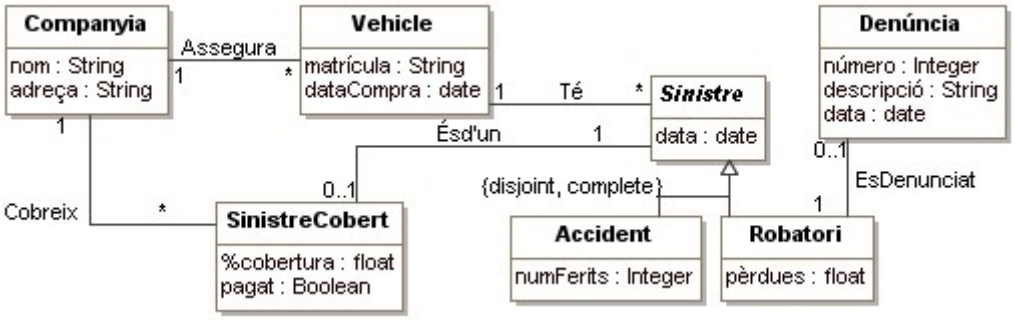
Solució

a. *En aquest cas es podrien utilitzar les dues alternatives. De totes maneres, si no hi ha ni atributs, ni altres operacions podríem utilitzar l'opció de l'interface.*
b. *S'hauria d'utilitzar la segona alternativa. Una interface no pot tenir operacions concretes i en aquest cas l'operació draw ha de ser concreta ja que té una part igual per a totes les seves subclases.*
c. *S'hauria d'utilitzar la segona alternativa. Una interface no pot tenir atributs privats que s'han d'aplicar a cada instància.*
d. *S'hauria d'utilitzar la primera alternativa. Tot i que UML permet herencia múltiple (una classe no pot heretar de dues clases) si volem utilitzar un*

llengutage de programació que no permeti aquest tipus d'herència hauríem d'utilitzar la primera alternativa ja que una classe pot heretar d'una altra classe i implementar diverses interfícies

Exercici 10

Un consorci de companyies d'assegurances ens ha demanat que li dissenyem una part d'un sistema software per gestionar els sinistres dels vehicles que tenen assegurats. Els vehicles s'identifiquen per matrícula i es guarda la seva data de compra i la companyia a la que estan assegurats. Un vehicle pot tenir un sinistre en una determinada data. Un sinistre pot estar cobert per una companyia asseguradora. Si ho està, se'n coneix el percentatge de cobertura i si ha estat pagat o no. A més, de tots els sinistres possibles, interessa guardar informació específica dels accidents (es vol conèixer el nombre de ferits, si n'hi ha) i dels robatoris (es vol saber l'import de les pèrdues i si han estat denunciats o no). Les denúncies s'identifiquen per un número i se'n coneix també la data en què es van fer efectives i la seva descripció. A continuació disposeu de l'especificació feta per a aquest sistema.



R.I. Textuals:
- Claus: (Companyia, nom); (Vehicle, matrícula); (Sinistre, matrícula, data); (Denúncia, número); (SinistreCobert, nom, matrícula, data)
- La companyia asseguradora que cobreix el sinistre ha de ser la companyia on està assegurat el vehicle.
- Els sinistres coberts que corresponen a robatoris només poden ser pagats si tenen la denúncia corresponent.
- La data de la denúncia ha de ser posterior a la data del sinistre.
- Altres restriccions no rellevants pel problema

context CapaDomini::llistarVehicles(nomComp:String, data:Date): SetVehicles on SetVehicles={matrícula}
pre *companyia-existeix*: la companyia *nomComp* existeix.
post result= conjunt de matrícules dels vehicles assegurats per la companyia *nomComp* que tenen algun sinistre en data posterior a *data* amb una cobertura superior al 50% i que són de tipus robatori denunciat o bé de tipus accident sense ferits.

context CapaDomini::canviCompanyiaAsseguradora(matr:String, novaComp:Companyia)
exc *sinistres-no-pagats*: el vehicle té sinistres coberts per la companyia actual que no han estat pagats.
post es dóna de baixa la instància de l'associació entre la companyia actual i el vehicle *matr*.
post s'eliminen tots els sinistres que ha tingut el vehicle *matr*, així com les denúncies, si en té, pel cas dels sinistres amb robatori.

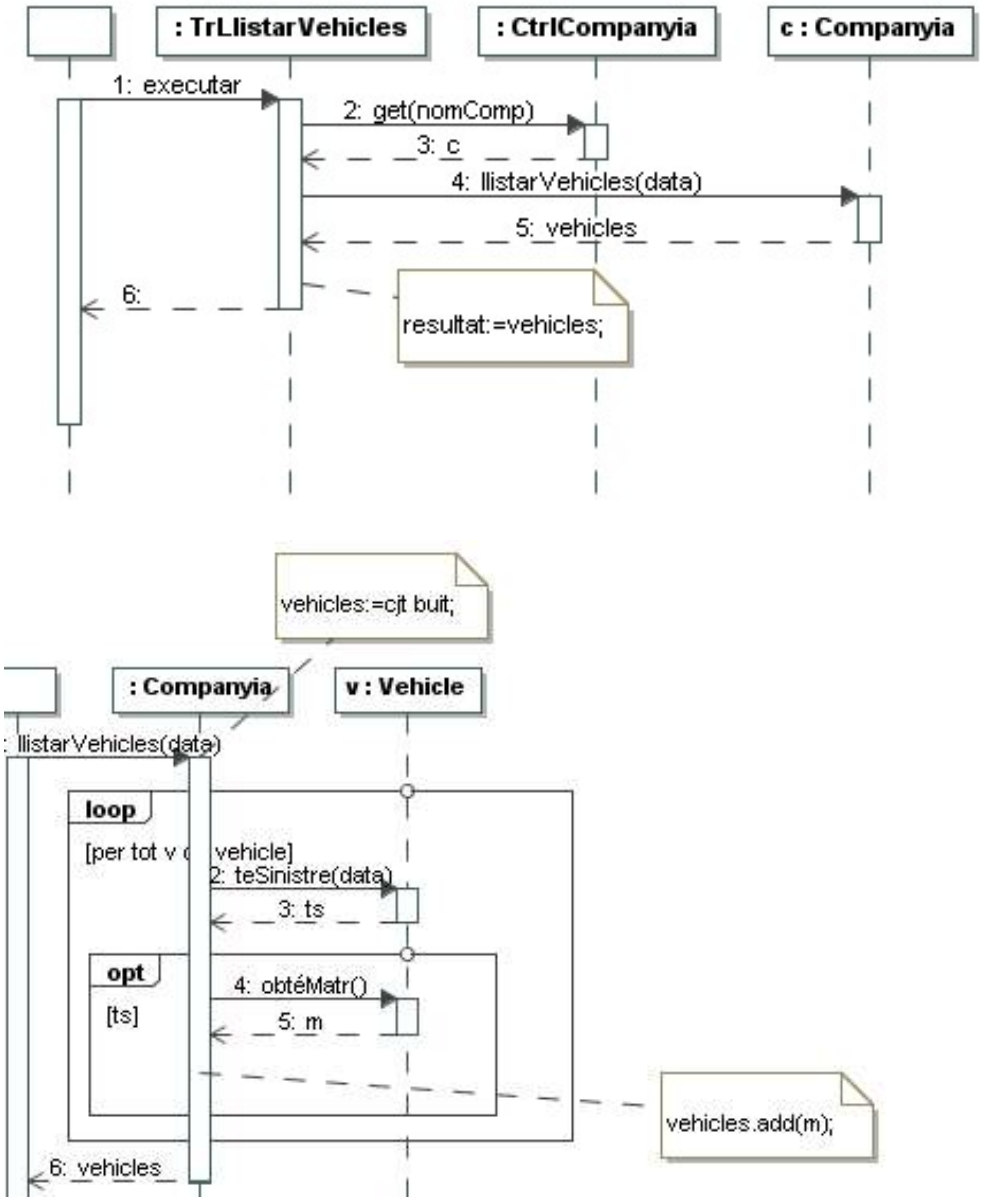
post s'eliminen totes les instàncies de sinistre cobert que fan referència a la companyia actual i al sinistre del vehicle *matr*.
post es dóna d'alta la instància de l'associació entre la companyia *novaComp* i el vehicle *matr*.

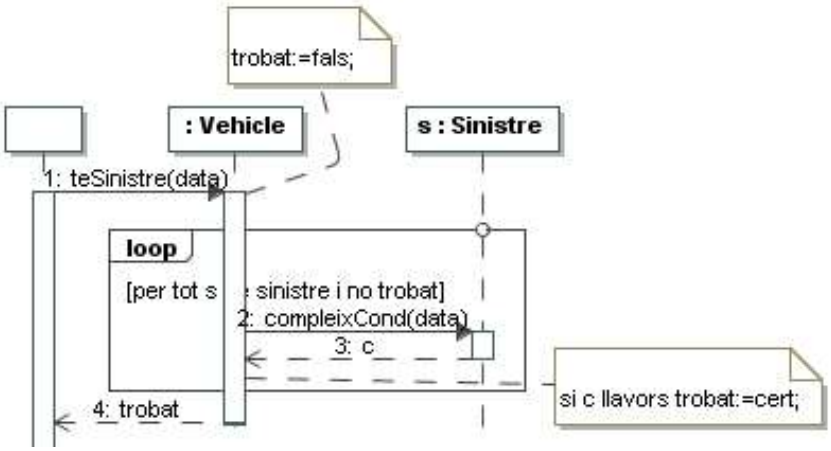
Es demana:

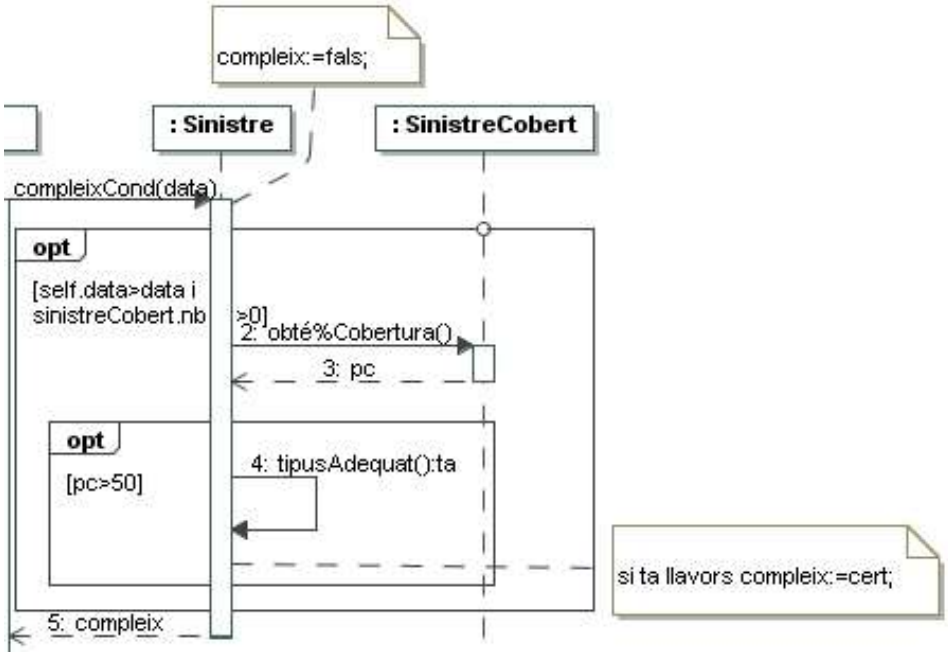
a) Diagrama de seqüència de l'operació *llistarVehicles* i *canviCompanyiaAsseguradora*. S'ha de justificar l'aplicació dels principis de disseny estudiats.
b) Diagrama de classes de la capa de domini.

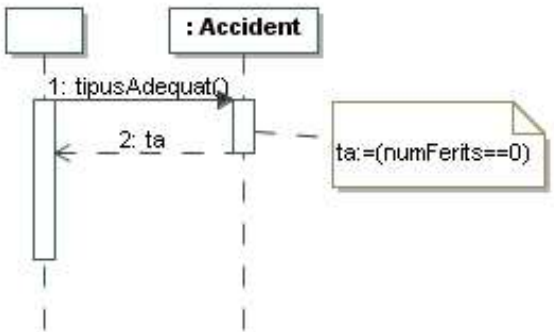
Solució

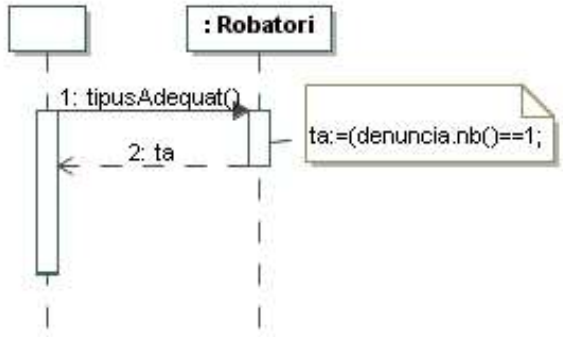
a) llistarVehicles



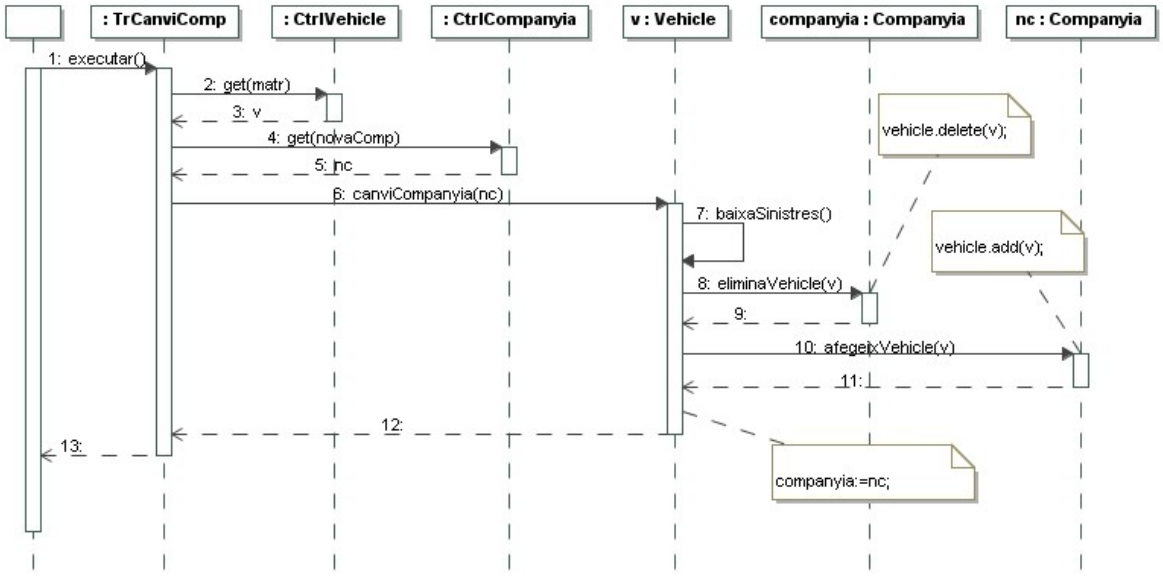

trobat:=fals;
: Vehicle
s : Sinistre
1: teSinistre(data)
loop
[per tot s sinistre i no trobat]
2: compleixCond(data)
3: c
si c llavors trobat:=cert;
4: trobat

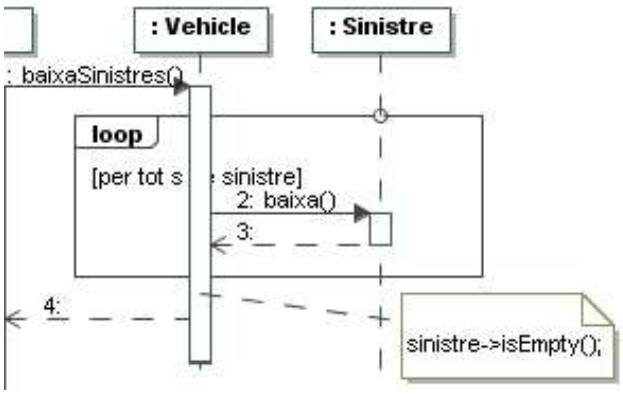

compleix:=fals;
: Sinistre
: SinistreCobert
compleixCond(data)
opt
[self.data>data i
sinistreCobert.nb
>0]
2: obté%Cobertura()
3: pc
opt
[pc>50]
4: tipusAdequat():ta
si ta llavors compleix:=cert;
5: compleix

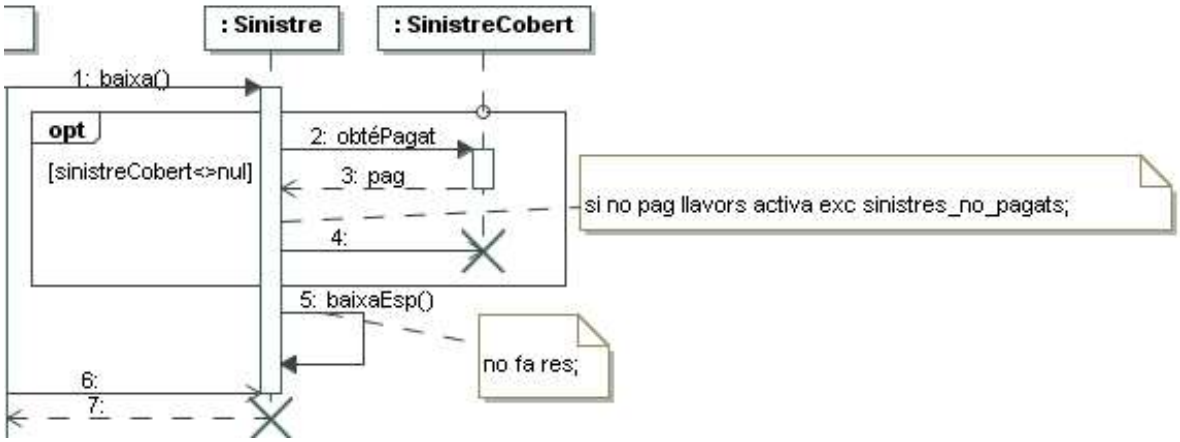

: Accident
1: tipusAdequat()
2: ta
ta:=(numFerits==0)

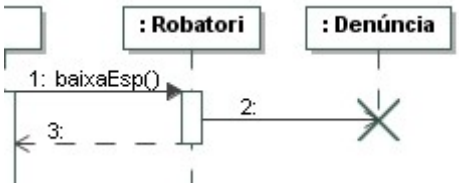


a) canviCompanyiaAsseguradora









b) Diagrama de classes de la capa de domini