

Capítol 3. Disseny de sistemes de programari

Autor: Xavier Franch. Revisores: Cristina Gómez, Lidia López



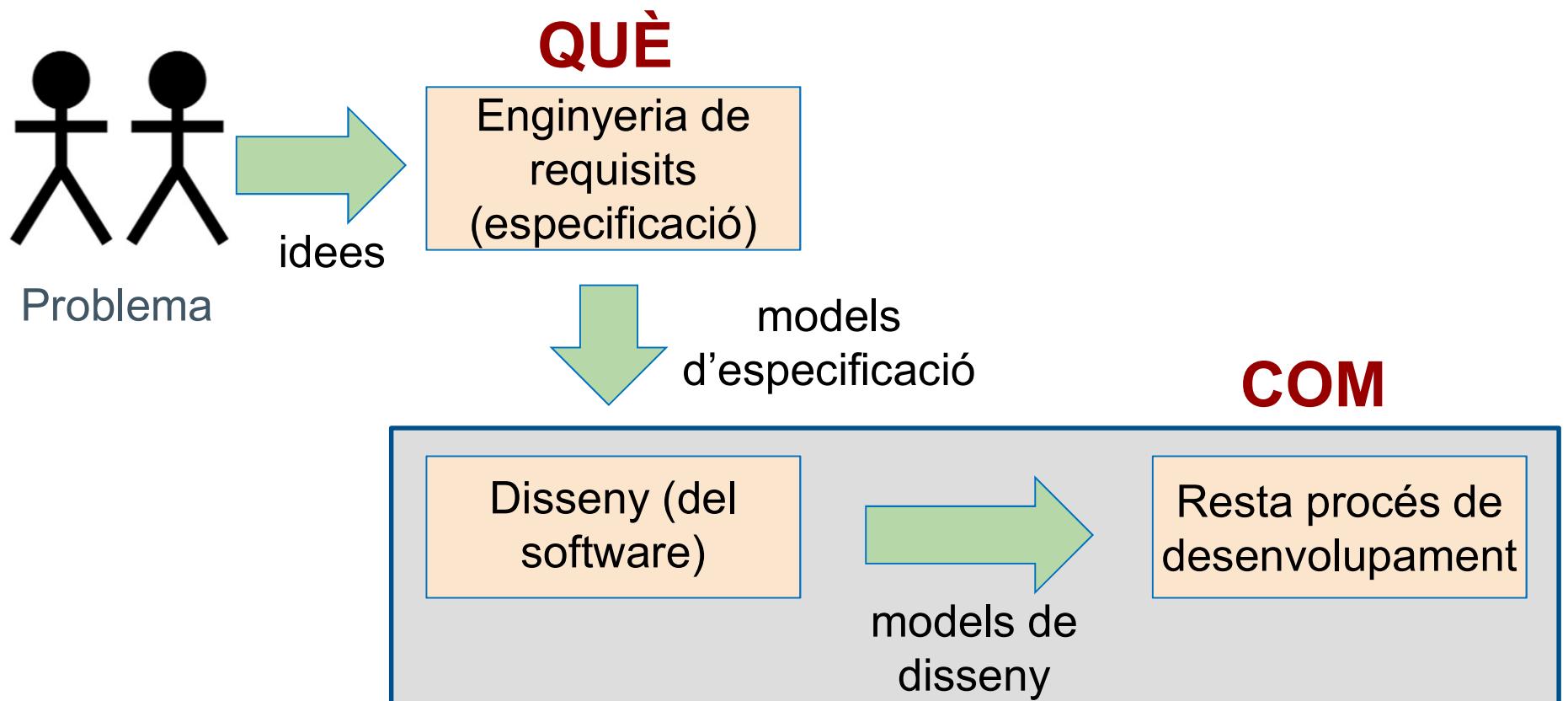
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Índex

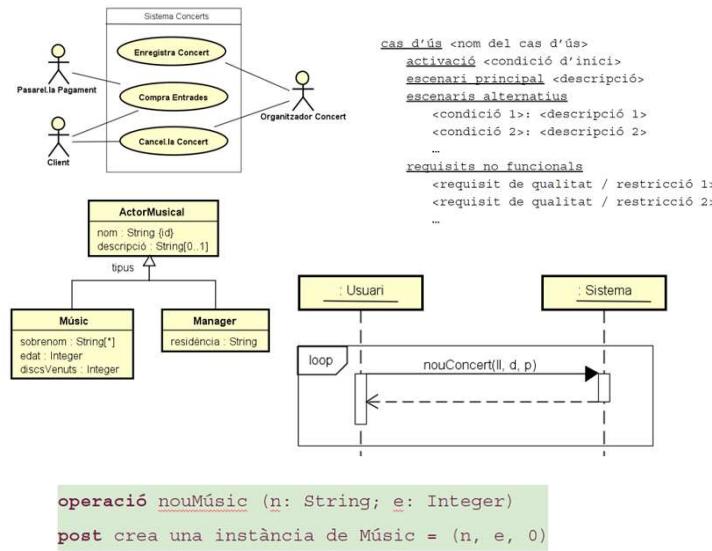
- 3.1 Introducció al disseny
- 3.2 Patrons arquitectònics
- 3.3 El procés de disseny
- 3.4 Disseny de la interfície d'usuari
- 3.5 Assignació de responsabilitats a capes
- 3.6 Simplificació del model
- 3.7 Disseny de les operacions

3.1 Introducció al disseny

Context

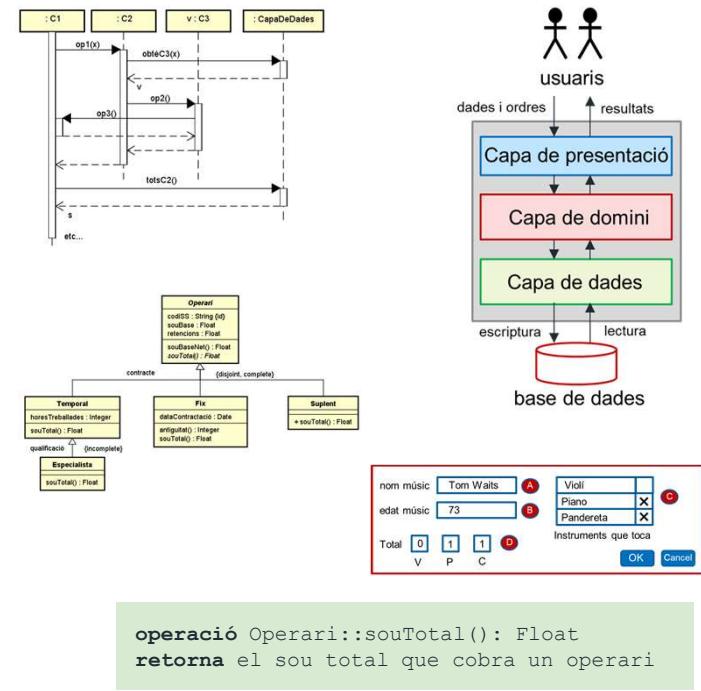


Models



Models
d'especificació

principis, mètodes
coneixement, tecnologia



Models de
disseny

Transició de l'especificació al disseny

Principis de disseny: regles que el nostre disseny ha de respectar

- en aquesta assignatura, seguirem principis universals, de la O.O., i certs requisits de qualitat (canviabilitat, eficiència, portabilitat, reusabilitat)

Mètodes: determinen el procés a aplicar

- en aquesta assignatura, usem la metodologia de Craig Larman

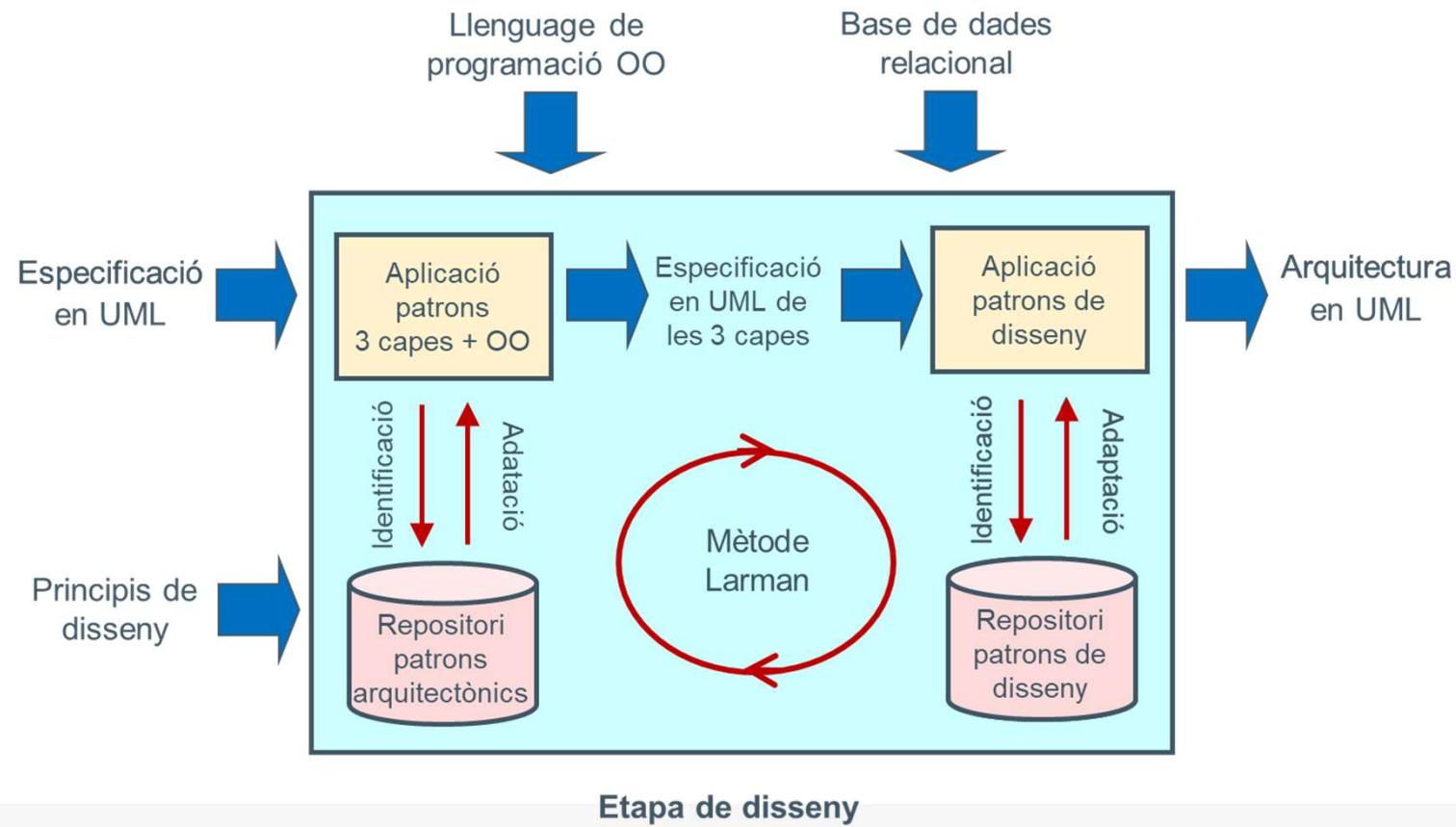
Coneixement: conjunt d'observacions i fets apresos al llarg dels anys

- en aquesta assignatura, pren la forma de **patrons**
 - **patrons arquitectònics**: defineixen l'estructura general del sistema
 - **patrons de disseny**: evolucionen el disseny cap a la seva forma final

Tecnologia: determina la forma final del resultat del disseny

- en aquesta assignatura, usem llenguatge de programació O.O. i base de dades relacional

Transició de l'especificació al disseny



3.2 Patrons arquitectònics

Context

Tot sistema software es regeix per una estructura global que determina:

- quins components hi existeixen
- com es comuniquen entre ells

La determinació d'aquesta estructura és clau i condiciona la resta del disseny

- una elecció incorrecte impactarà de forma negativa en el sistema resultant

Amb el pas dels anys, han anat emergint una col·lecció d'estructures de resultats contrastats

- aquestes estructures s'anomenen **patrons arquitectònics**
- la seva selecció depèn de diversos factors, relacionats sobretot amb els principis de disseny que es volen afavorir i amb el tipus de tecnologia amb què s'implementarà el sistema

Patrons arquitectònics més populars

En aquesta assignatura, apliquem dos patrons arquitectònics populars:

- **Patró arquitectònic en capes**: assignació de responsabilitats a les diverses capes definides
 - en aquesta assignatura, tindrem tres capes
- **Patró arquitectònic d'orientació a objectes (O.O)**: el sistema es veu com una col.lecció d'objectes que col.laboren
 - no és cap novetat...

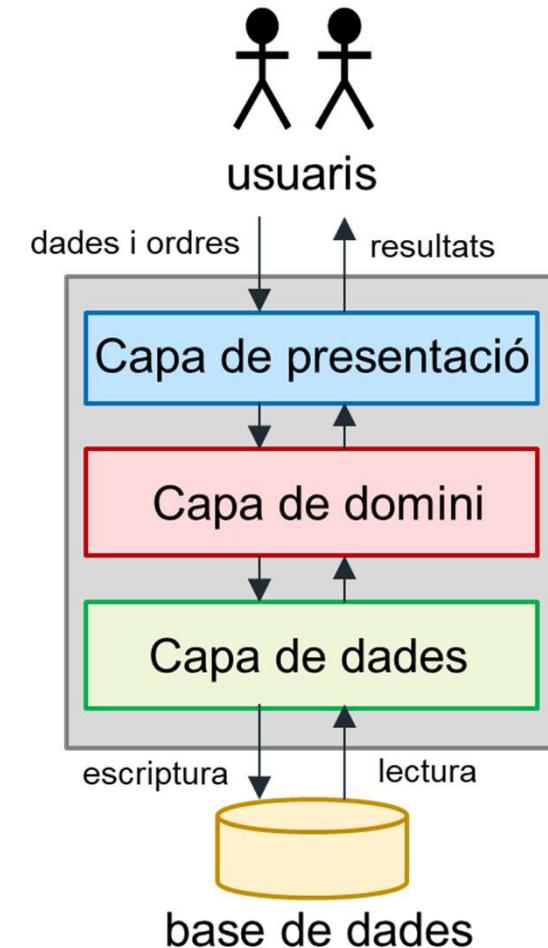
Altres patrons arquitectònics:

- arquitectura de serveis (SOA)
- arquitectura hexagonal
- ...

Patró arquitectònic en tres capes

El sistema s'estructura en tres capes que es comuniquen entre elles:

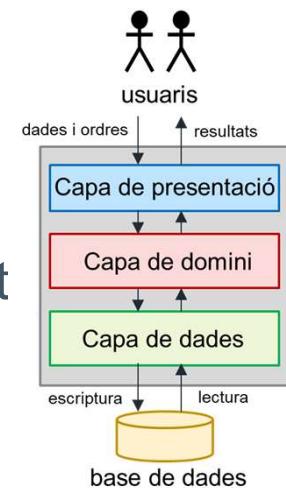
- **Capa de presentació**
 - s'ocupa d'interaccionar amb els usuaris
- **Capa de domini**
 - implementa les funcionalitats del sistema
- **Capa de dades**
 - assegura la persistència de les dades



Patró arquitectònic en tres capes i principis de disseny

El patró en capes afavoreix dos *principis de disseny universals*:

- **Cohesió alta:** un element de disseny (una capa, una classe, ...) és cohesiu si té un únic objectiu ben definit
 - cada capa té una missió ben definida (v. transpa anterior)
- **Acoblament baix:** un element de disseny té baix acoblament si evita interaccions innecessàries amb altres elements
 - la capa de presentació no es comunica directament amb la capa de dades



També afavoreix altres principis en forma de requisits de qualitat:

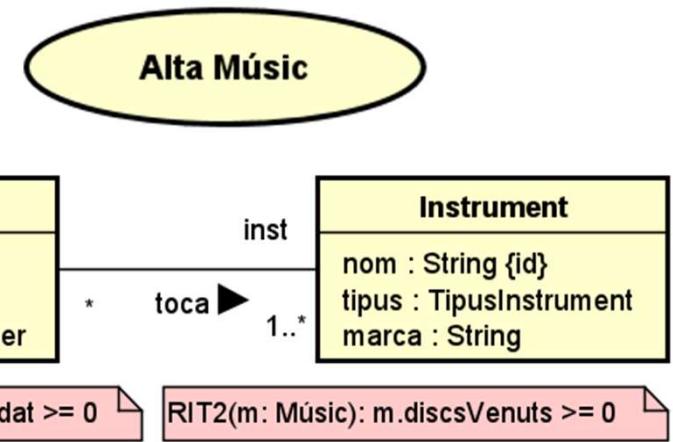
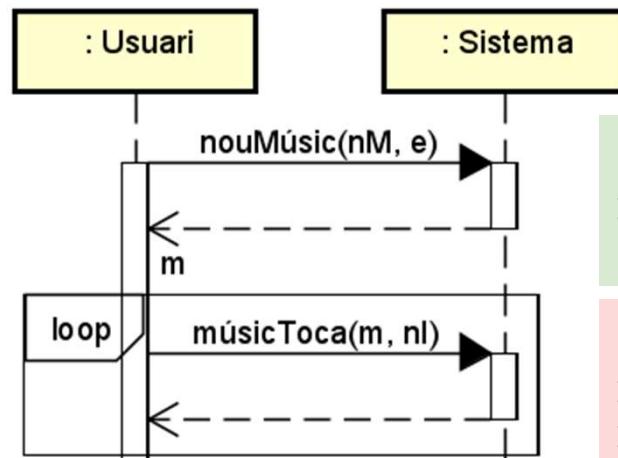
- **canviabilitat:** els canvis interns en una capa no haurien d'affectar les altres
- **portabilitat:** canvis en la tecnologia de la base de dades o els elements de la interfície gràfica no haurien d'affectar la capa de domini

Mecànica del patró en tres capes: exemple, especificació

cas d'ús Alta Músic

activació L'Usuari vol donar d'alta un nou músic

escenari principal L'Usuari entra el nom i edat del músic i tot seguit, un a un, els noms dels instruments que el músic toca



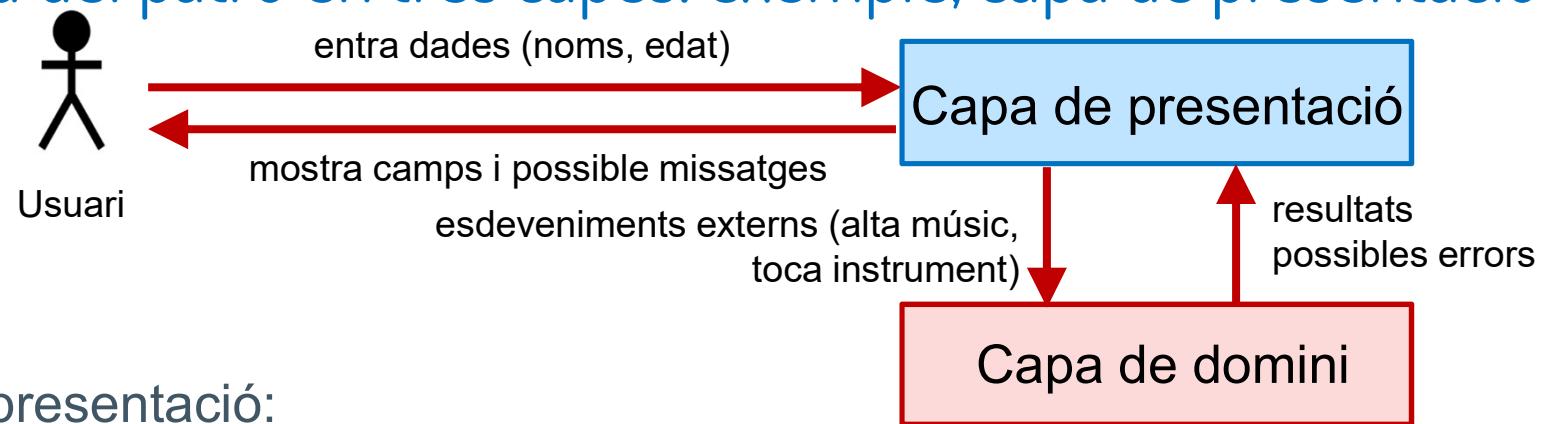
RIT1(m: Músic): m.edat >= 0

RIT2(m: Músic): m.discsVenuts >= 0

operació nouMúsic (nM: String; e: Integer): Músic
post crea una instància m de Músic = (nM, e, 0)
retorna m

operació músicToca (m: Músic; ni: String)
pre existeix al sistema un Instrument x amb nom ni
post s'enregistra que m toca x

Mecànica del patró en tres capes: exemple, capa de presentació



La capa de presentació:

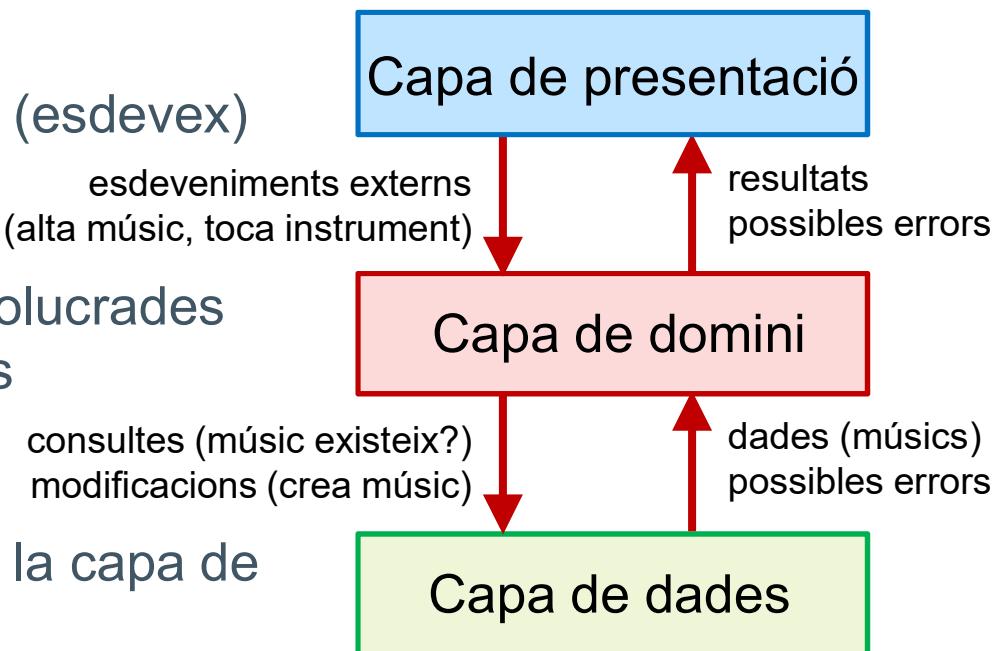
- gestiona la interfície d'usuari
- recull peticions d'usuari (**esdeveniments de presentació**)
- ordena peticions d'accions a la capa de domini
- recull resultats de la capa de domini
- presenta resultats a l'usuari

La capa de presentació sap com comunicar-se amb l'usuari, però desconeix com respondre les seves peticions

Mecànica del patró en tres capes; exemple, capa de domini

La capa de domini:

- detecta esdeveniments externs (esdevex)
- controla la seqüencialització adequada d'aquests esdevex i la correctesa de les dades involucrades
- ordena els canvis d'estat exigits per les operacions
- elabora els resultats demanats per les consultes i les retorna a la capa de presentació



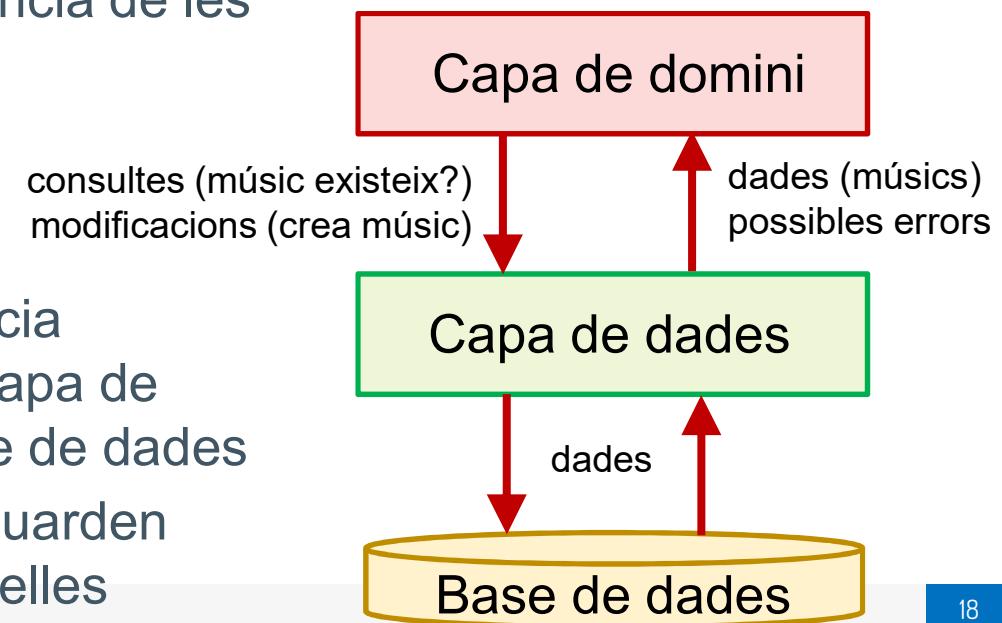
La capa de domini sap com resoldre les necessitats de l'usuari, però desconeix on es guarden les dades i com es presenten a l'usuari

Mecànica del patró en tres capes; exemple, capa de dades

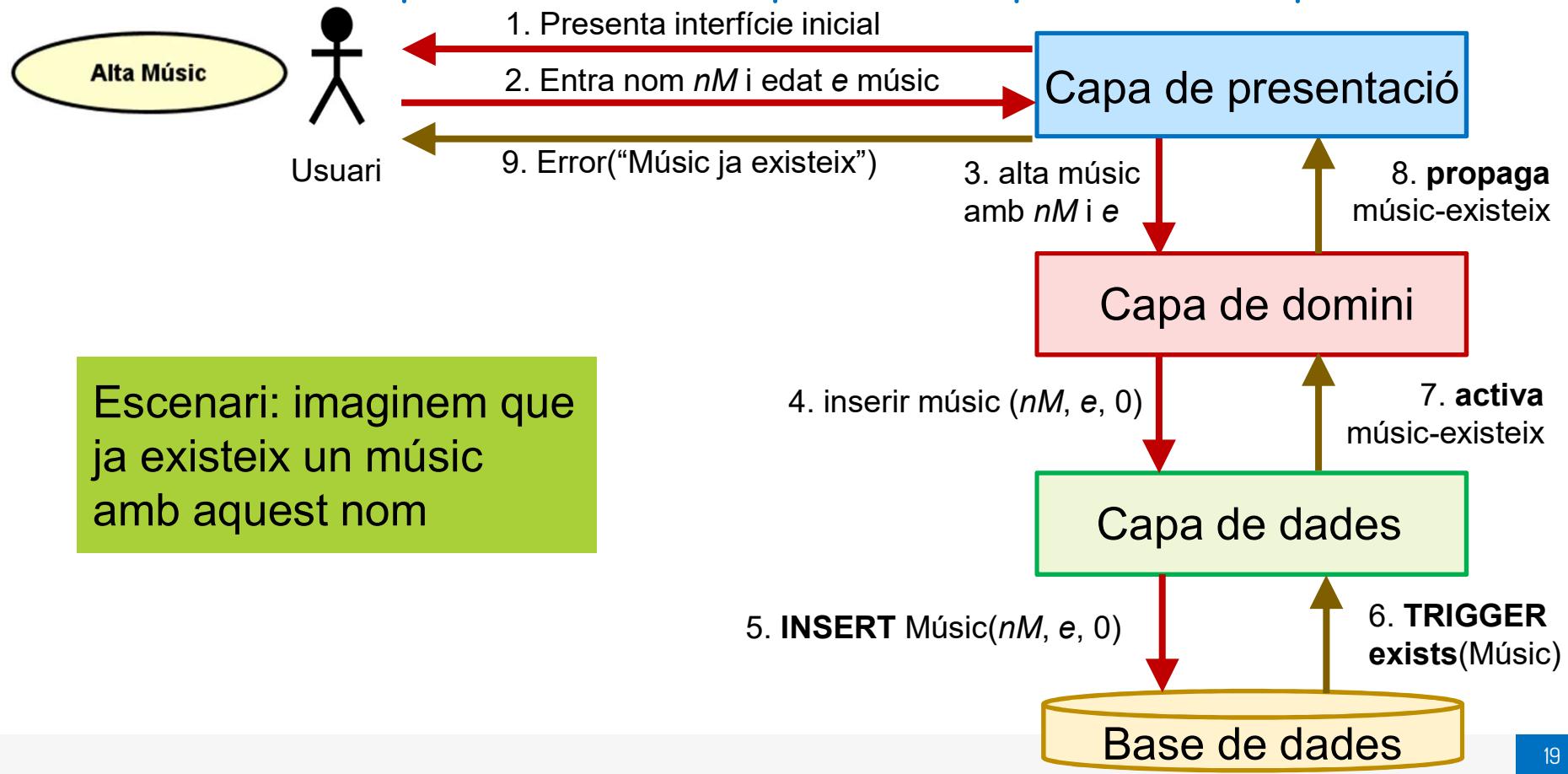
La capa de dades:

- defineix la forma en què s'emmagatzemaran les dades en la base de dades
- s'ocupa d'assegurar la persistència de les dades
- ofereix una col·lecció versàtil d'operacions, la forma de les quals depèn de l'estrategia concreta d'abordar la persistència
- en tot cas, rep peticions de la capa de domini i les transporta a la base de dades

La capa de dades sap com i on es guarden les dades, però no sap què fer amb elles



Mecànica del patró en tres capes: exemple, resum (parcial)



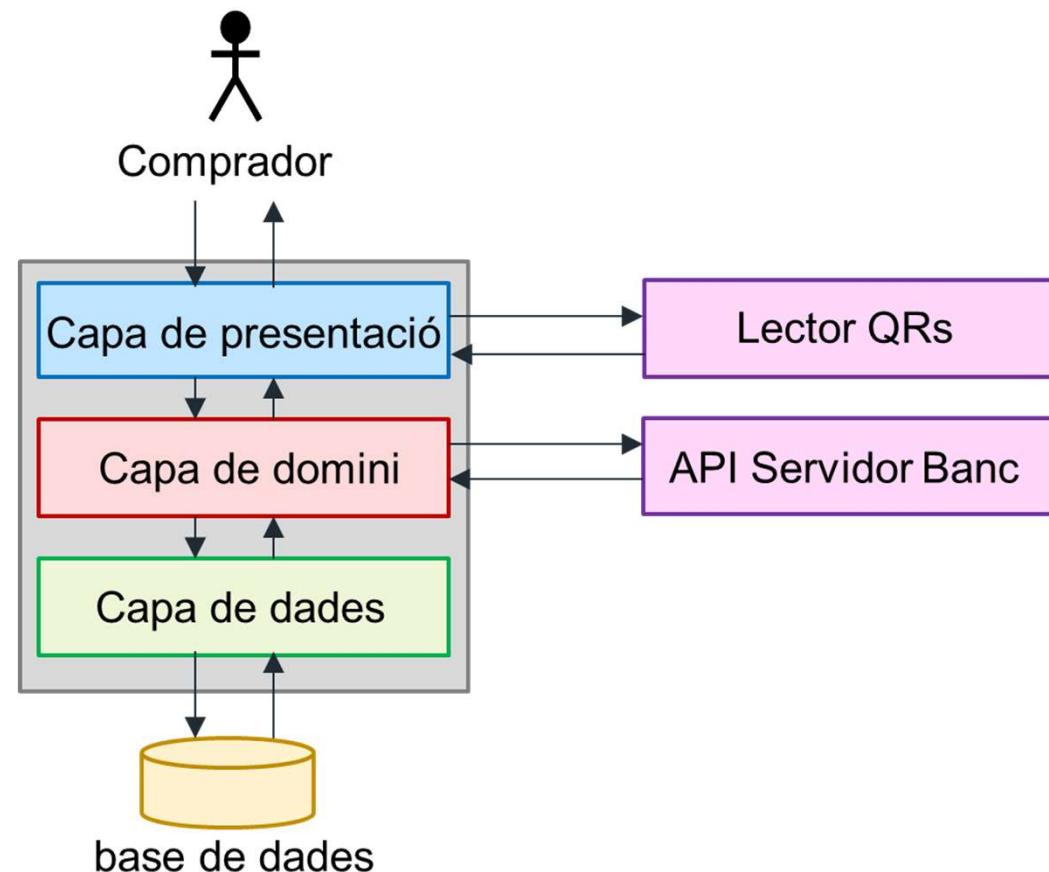
Extensió: comunicació amb altres sistemes o dispositius

Sovint, el nostre sistema ha de comunicar-se amb altres sistemes o fins i tot dispositius per assolir els seus objectius

- aquesta comunicació pot ocórrer a qualsevol nivell, p.e.
 - la capa de presentació pot necessitar accedir a una web remota
 - la capa de domini pot precisar comunicar-se amb un sistema diferent per processar una funcionalitat
 - la capa de domini pot necessitar una mesura proporcionada per un dispositiu físic
 - la capa de dades pot treballar sobre diverses bases de dades
- en tots aquests casos, el nostre sistema necessita invocar operacions ofertes per aquests altres sistemes / dispositius
 - mitjançant un servei o una API (Application Programming Interface)

Extensió: exemple

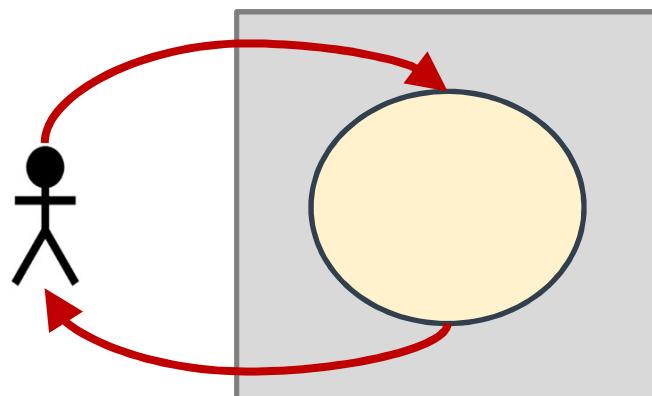
Sigui el cas d'ús Assistència Concert que s'activa quan una persona vol entrar a un concert a punt de celebrar-se. El sistema ha de reconèixer el codi QR de l'entrada i també ha de permetre comprar l'entrada al moment usant una tarja de crèdit



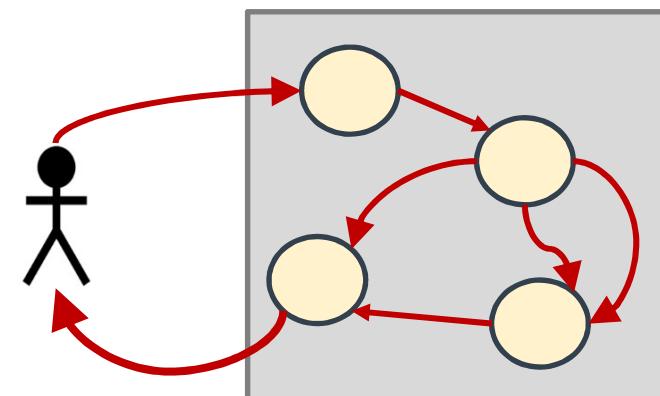
Patró arquitectònic orientat a objectes (O.O.)

El sistema consta d'una col·lecció d'objectes que interaccionen i col·laboren per complir l'especificació

- Evolució de la visió de l'especificació



Especificació

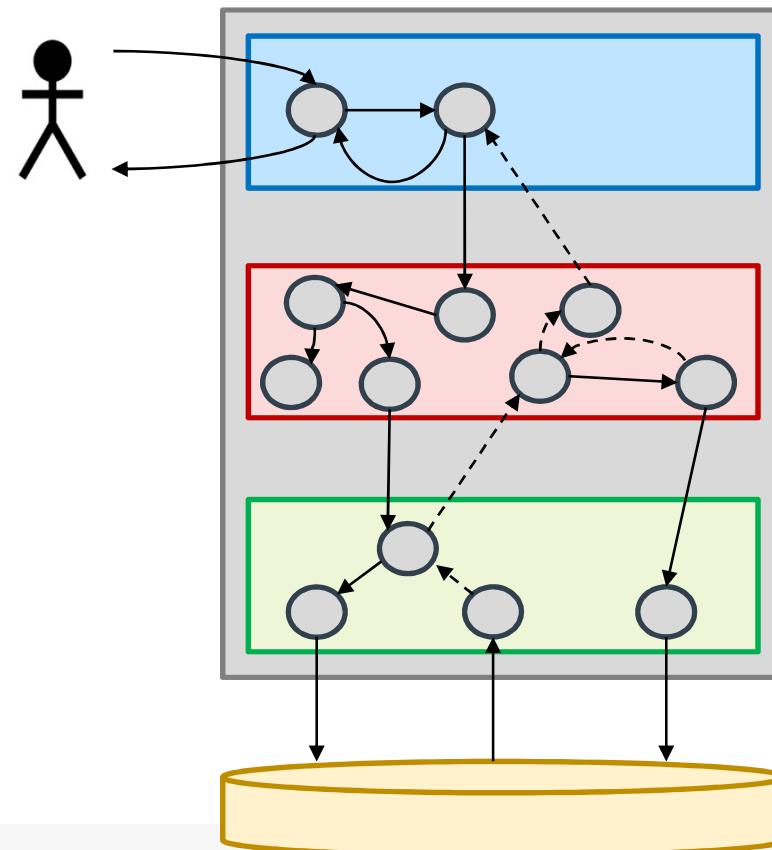


Disseny

Combinació del patró en tres capes i el patró 0.0.

Cada capa del sistema es representa mitjançant una col·lecció d'objectes que col·laboren

Les capes es comuniquen mitjançant alguns d'aquests objectes



Altres consideracions del patró 0.0.

En comparació amb l'especificació, els models OO de disseny hauran de detallar altres temes d'interès per als desenvolupadors; entre d'altres:

- En els diagrames de classe: informació més completa
 - s'hi introdueixen operacions
 - visibilitat, àmbit, navegabilitat, ...
- En els diagrames de seqüència: com s'implementen les funcionalitats
 - col·laboracions entre objectes
 - cal crear i destruir objectes
- En els contractes: elaboració de les precondicions
 - es donen per fetes o cal comprovar-les?
 - desapareix el concepte de redundància (contractes auto-continguts)

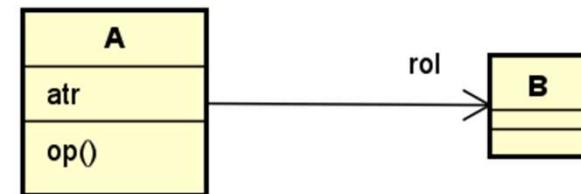
Patró 0.0.: Model de dades – visibilitat

La **visibilitat** defineix quins objectes “veuen” (i.e., tenen dret a consultar i eventualment modificar) informació declarada en un diagrama de classes

- Afecta principis de disseny com ara la canviabilitat i la portabilitat

La visibilitat es pot establir sobre tres tipus d'elements:

- Atributs
- Operacions
- Rols (extrems d'associació) *navegables*



Donat l'element v definit a la classe A , UML defineix tres visibilitats:

- Públic (+): qualsevol classe que veu A , veu v
- Privat (-): x només és visible des de A (cas per defecte)
- Protegit (#): x és visible des de A i des de tots els descendents de A (si n'hi ha)
 - en aquesta assignatura, no usarem aquesta visibilitat

Patró 0.0.: Model de dades – àmbit

L’**àmbit** determina si un element de disseny és aplicables a objectes individuals o a la classe que defineix l’element

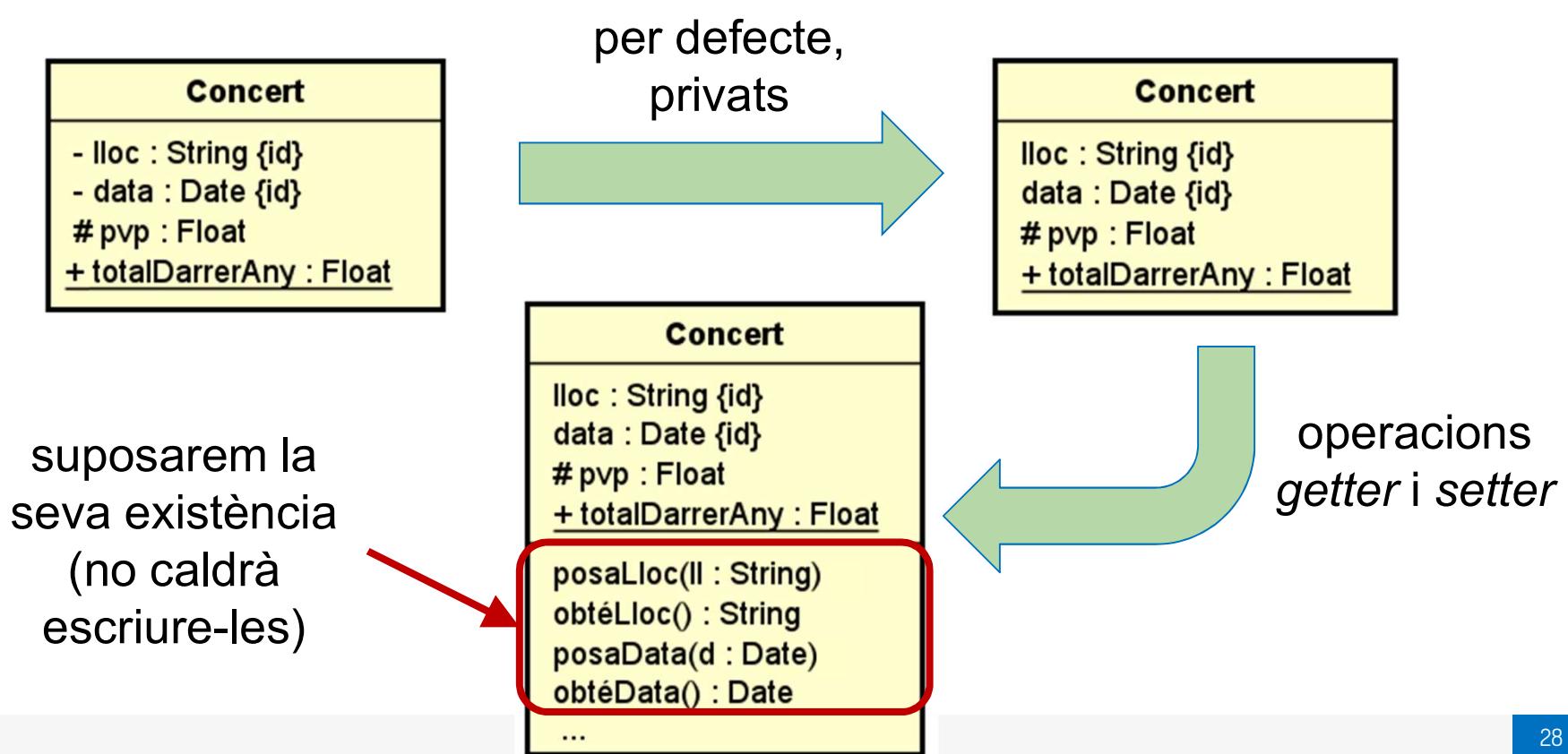
L’àmbit es pot establir sobre dos tipus d’elements:

- Atributs
- Operacions

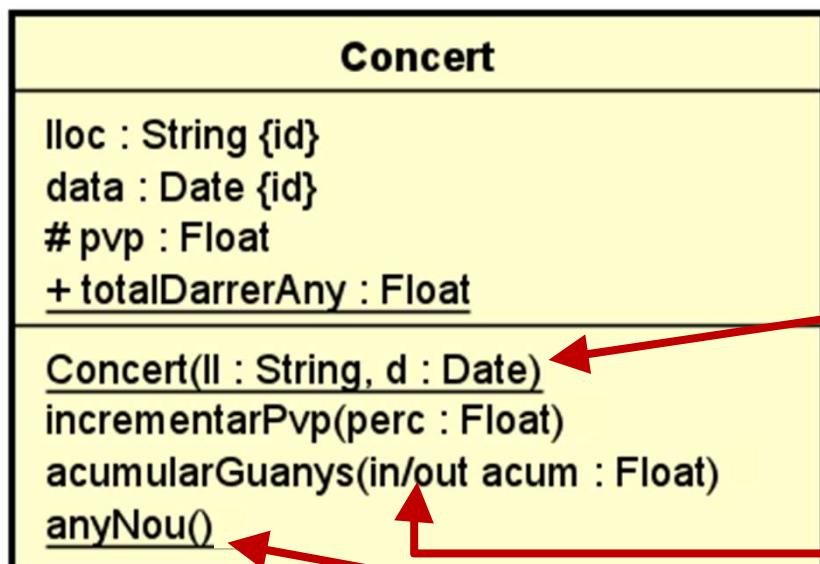
Donat l’element v definit a la classe A , UML defineix dos àmbits:

- D’instància (no estàtic): v està associat als objectes de A
 - cas per defecte
 - referència: $obj.v$, essent obj una instància de la classe A
- De classe (estàtic): v està associat a A
 - referència: $A.v$

Patró 0.0.: Model de dades – cas dels atributs



Patró 0.0: Model de dades – cas de les operacions



més *setters* i
getters...

per defecte, públiques!

operació de classe (estàtica)
per a la creació d'una
instància de classe Concert
(clau com a paràmetre,
opcionalment altres camps)

els paràmetres poden ser **in**
(per defecte), **out** o **in/out**
operació de classe (estàtica)

Patró 0.0.: Model de comportament – Contractes associats

inicialitza tots els atributs de classe



operació Concert(ll: String; d: Date)
post crea un objecte c = (ll, d, 0) de classe Concert

Concert
lloc : String {id}
data : Date {id}
pvp : Float
+ totalDarrerAny : Float
Concert(ll : String, d : Date)
incrementarPvp(perc : Float)
acumularGuanys(in/out acum : Float)
anyNou()

referència a l'objecte que rep la invocació (opcional)

operació de classe modifica atribut de classe

operació acumularGuanys(**in/out** acum: Float)
post acum += totalDarrerAny

operació Concert(ll: String; d: Date)

post crea un objecte c = (ll, d, 0) de classe Concert

operació incrementarPvp(perc: Float)
pre perc > 0
post self.pvp += self.pvp*perc

operació anyNou()
post totalDarrerAny = 0

modificació d'atribut d'entrada / sortida

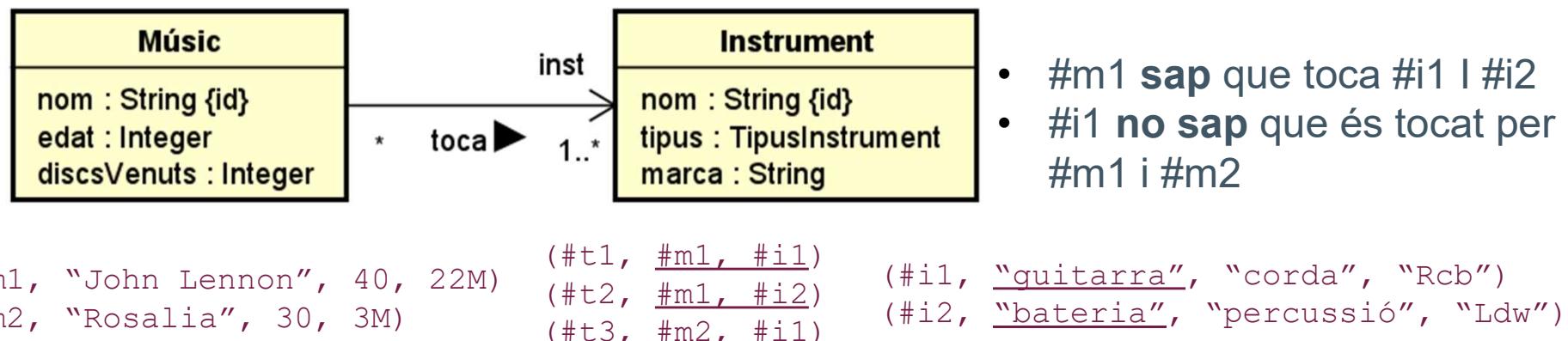


operació acumularGuanys(**in/out** acum: Float)
post acum += totalDarrerAny

Patró 0.0.: Model de dades – navegabilitat

La **navegabilitat** estableix si és possible anar d'una classe *A* cap a una altra classe *B* resseguint una associació ass usant el rol *r*

- Si *A* es navegable cap a *B* resseguint ass, un objecte d'*A* pot saber amb quins objectes de *B* està relacionat mitjançant ass usant *r*
- Afecta principis de disseny com ara l'acoblament, la canviabilitat, la portabilitat i la usabilitat

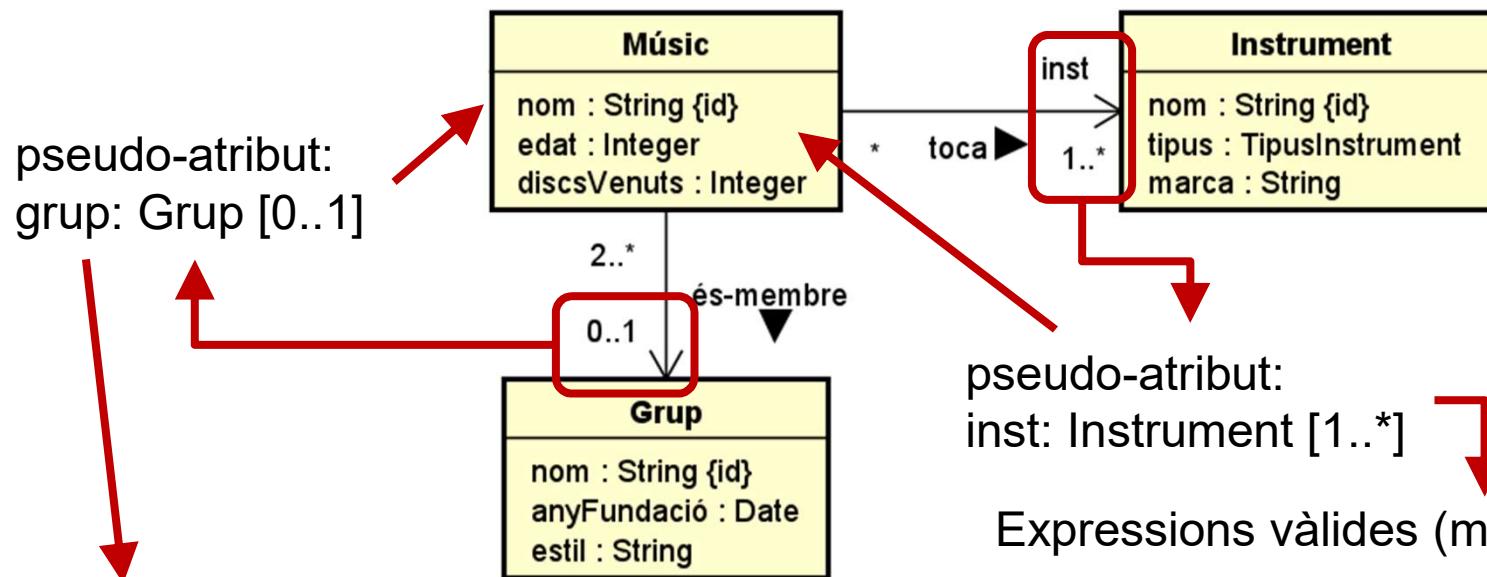


Patró 0.0.: Model de dades – accés als rols (1)

Sigui una classe A associada amb una altra classe B mitjançant una associació ass navegable de A a B

- un objecte v de classe A té accés a tots els objectes de classe B associats ass usant el nom de rol r
 - si no té nom explícit, usem el nom de la classe B amb la primera lletra minúscula (b)
- es diu que r és un **pseudo-atribut** de la classe A
 - no és un atribut, però es comporta com un atribut
 - p.e., es pot modificar i consultar el seu valor
 - p.e., per defecte, els rols són privats
 - el seu tipus (implícit) depèn de la multiplicitat de r

Patró 0.0.: Model de dades – accés als rols (2)



pseudo-atribut:
grup: Grup [0..1]

pseudo-atribut:
inst: Instrument [1..*]

Expressions vàlides (m: Músic, g: Grup):

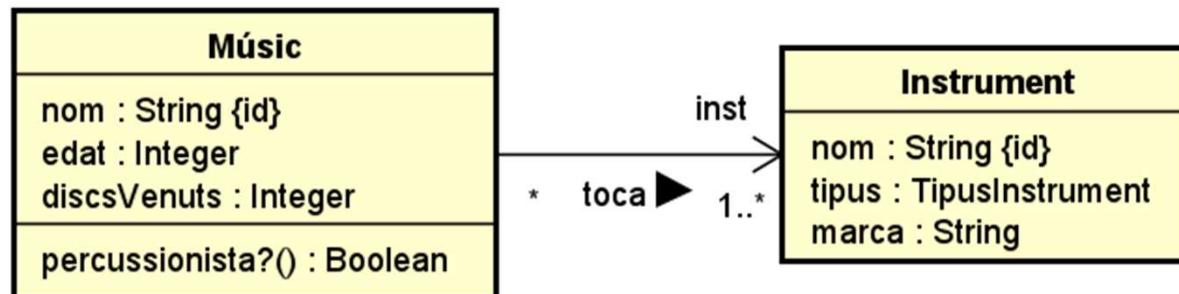
- `m.grup = g`
- `si m.grup == g`
- `si m.grup no és nul`

Expressions vàlides (m: Músic, v: Instrument):

- `m.inst.afegeix(v)`
- `afegeix v a m.inst`
- `m.inst.esborra(v); esborra v de m.inst`
- `si v dins de m.inst`

} dos estils

Patró 0.0.: Model de comportament – Disseny



operació `percussionista?() : Boolean`

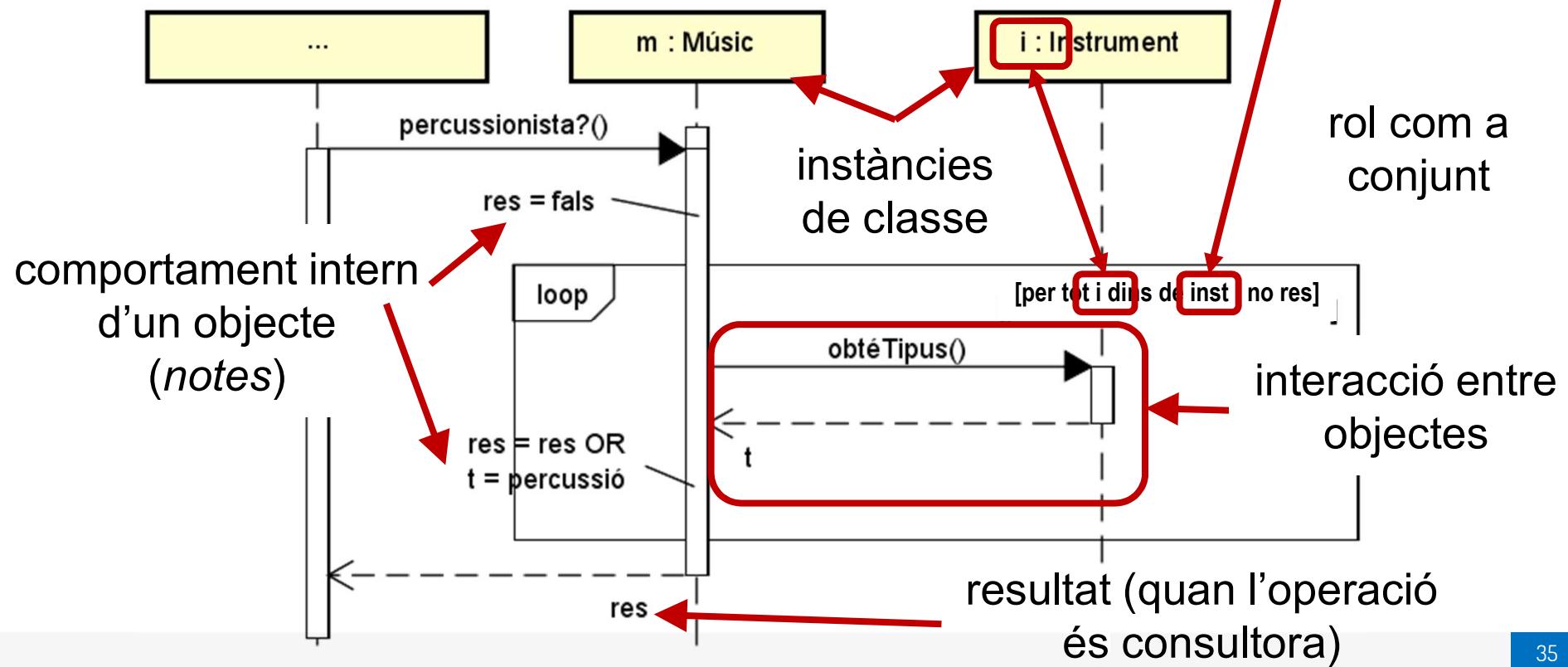
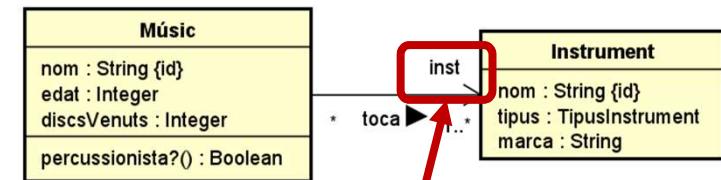
retorna cert si self toca algun instrument de tipus percussió

Preguntes:

- D'on surten els contractes?
- Com es dissenya cada operació a partir del contracte?

Calen diagrames de seqüència

Patró 0.0.: Diagrames de seqüència



Patró 0.0.: Creació d'objectes



operació Músic(n: String, e: Integer) Opció 1
post crea un nou Músic m = (n, e, 0)

operació Músic(n: String, e: Integer, sI: Conjunt(String))

Opció 2

pre sI no és buit

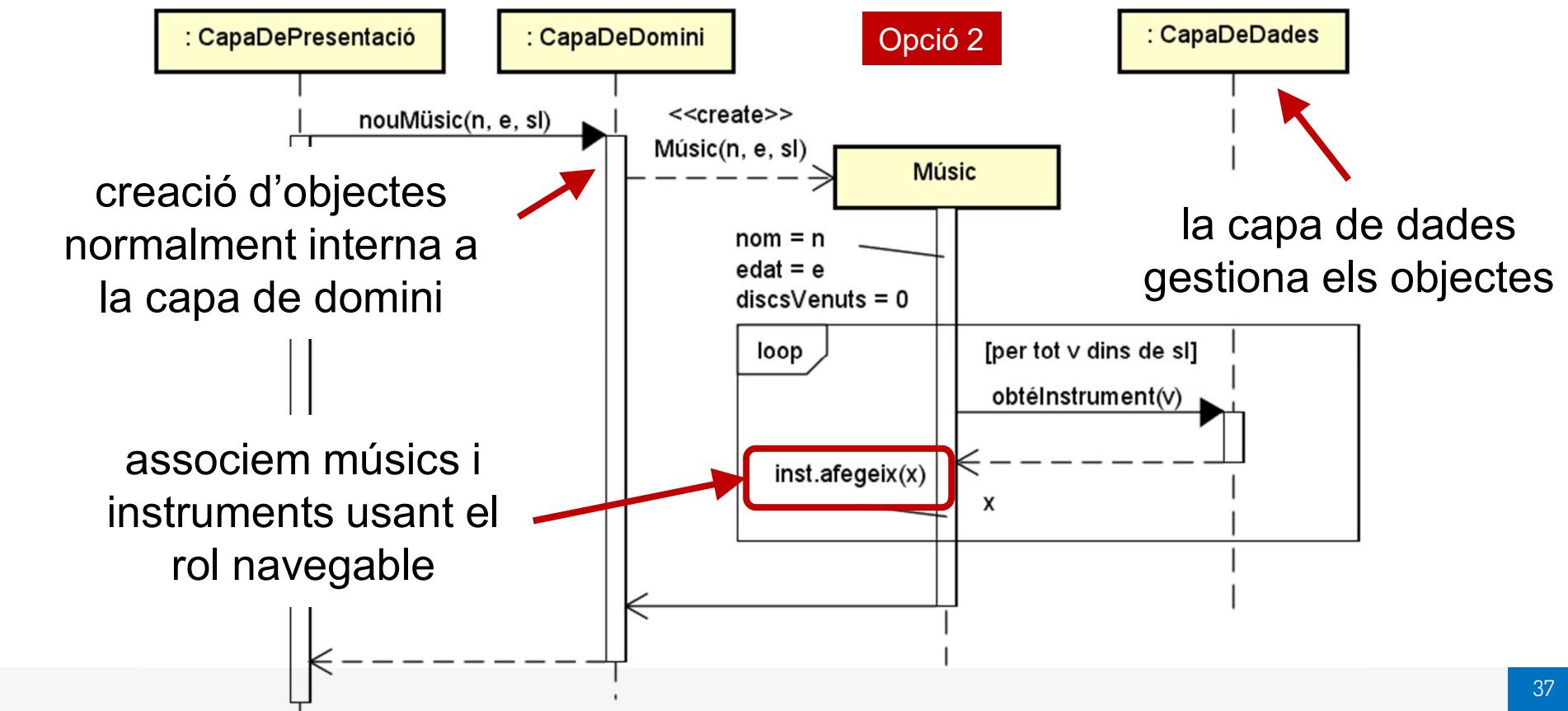
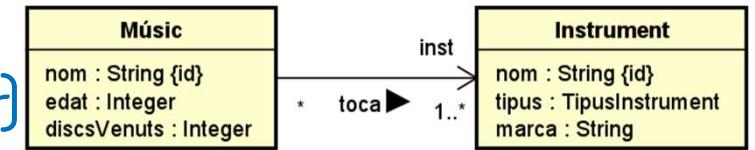
pre per tot v dins de sI: existeix un instrument x tal que x.nom = v

post crea un nou Músic m = (n, e, 0)

post per tot v dins de sI: associa m amb l'Instrument x tal que x.nom = v

L'elecció d'una o altra opció depèn del cas d'ús

Patró 0.0.: Creació d'objectes (*spoiler*)



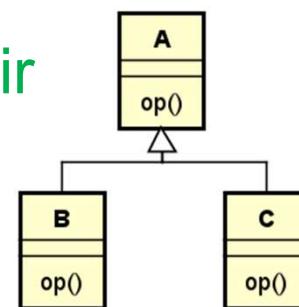
Patró 0.0.: Polimorfisme

“the property of crystallizing in two or more forms with distinct structure”

– diccionari Merriam-Webster

En el context dels diagrames de classe, el **polimorfisme** és una propietat que *poden* exhibir les operacions d'una jerarquia d'especialització

- si A és antecessor de B i C, A pot **declarar/definir** una operació op que es comporti de manera diferent en B i/o en C
 - op és una **operació polimòrfica**

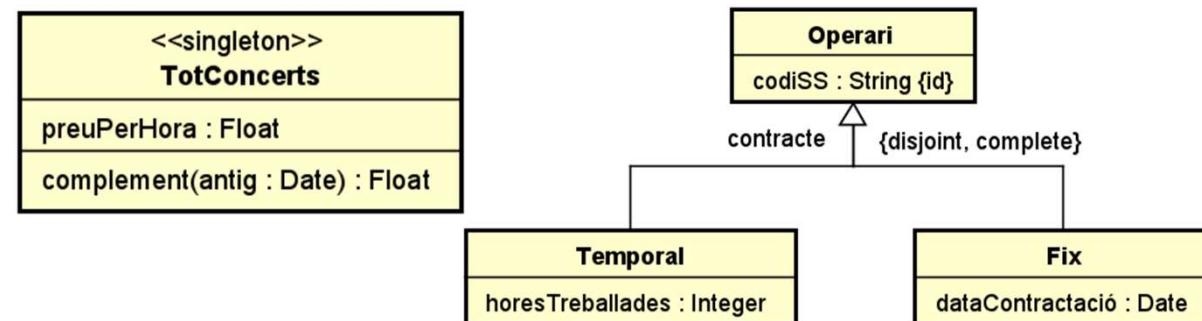


Polimorfisme, exemple: enunciat

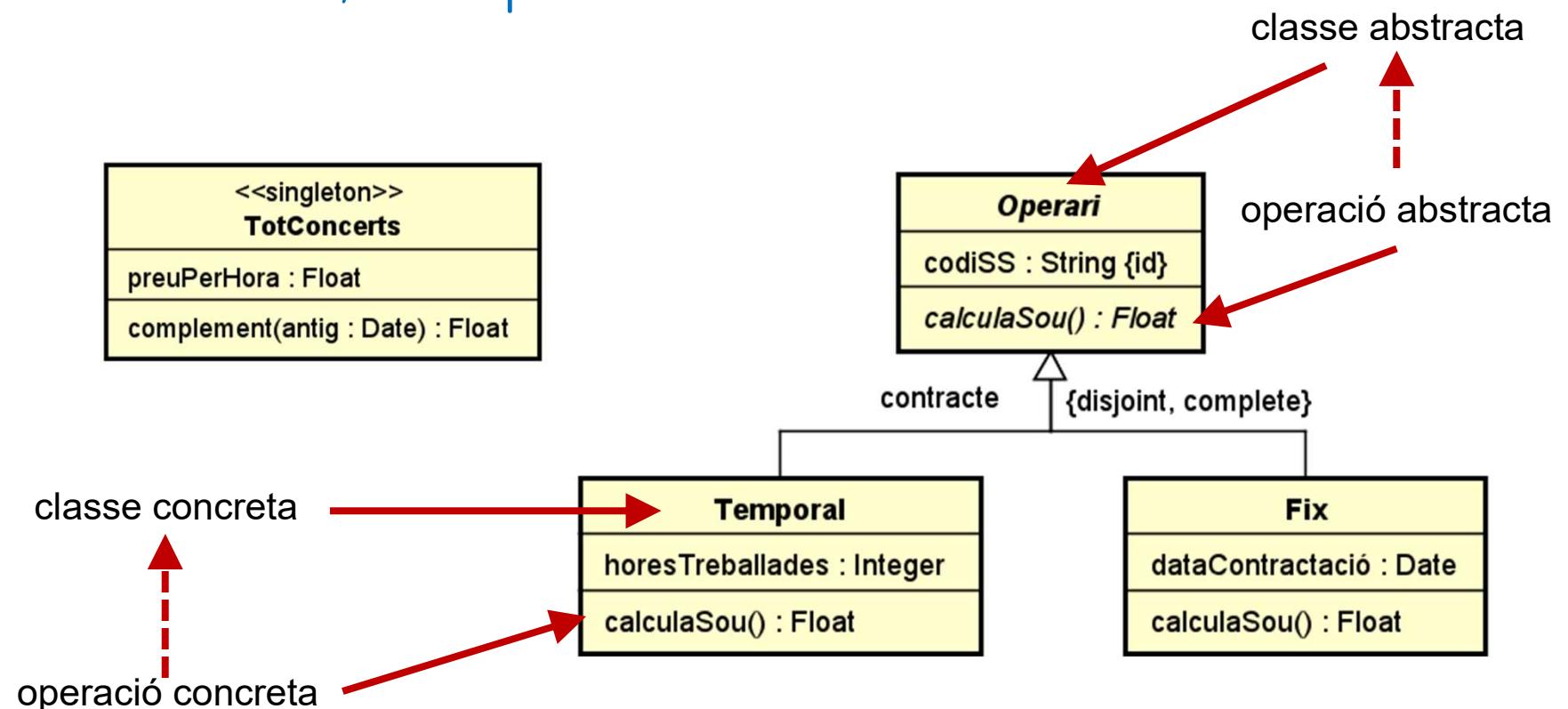
Volem una operació per saber el sou mensual dels operaris que treballen en l'organització de concerts que depèn del tipus d'operari:

- Per als operaris amb contracte fix, el sou es calcula a partir d'un complement d'antiguitat
- Per als operaris amb contracte temporal, el sou es calcula a partir de les hores treballades

Volem fer el model de comportament d'aquesta operació, usant el diagrama de classes de la dreta



Polimorfisme, exemple: model de dades



Polimorfisme: Definicions i característiques

Operació abstracta

- Operació de la qual només se'n declara la signatura
- No es pot executar
- Sintàcticament, s'escriu en cursiva (o escrivim { abs } al costat)

Classe abstracta

- Classe que conté alguna operació abstracta
- No se'n poden instanciar objectes
- Sintàcticament, s'escriu en cursiva

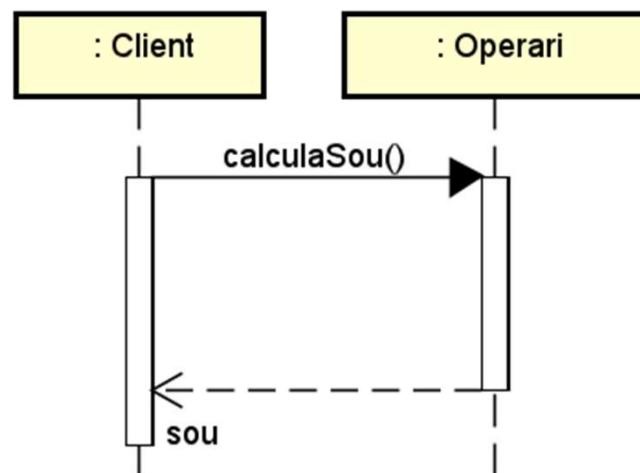
Operació concreta

- Operació que té un comportament associat
- Es pot executar
- Es pot **redefinir** en una classe filla en la mateixa jerarquia

Classe concreta

- Classe que no conté cap operació abstracta
- Se'n poden instanciar objectes

Polimorfisme, exemple: model del comportament (1)



notació UML: sempre es pot prefixar el nom d'un element de classe, amb el nom de la classe

operació Operari::calculaSou(): Float
retorna el sou total que cobra un operari

Vinculació dinàmica: Quan el Client invoca calculaSou sobre un objecte de classe Operari, la determinació de quina operació s'executa es fa en temps d'execució, dependent de la subclasse a la que pertany l'objecte

Polimorfisme, exemple: model del comportament (2)

Les diverses definicions de l'operació polimòrfica han de tenir el seu propi contracte i diagrama de seqüència

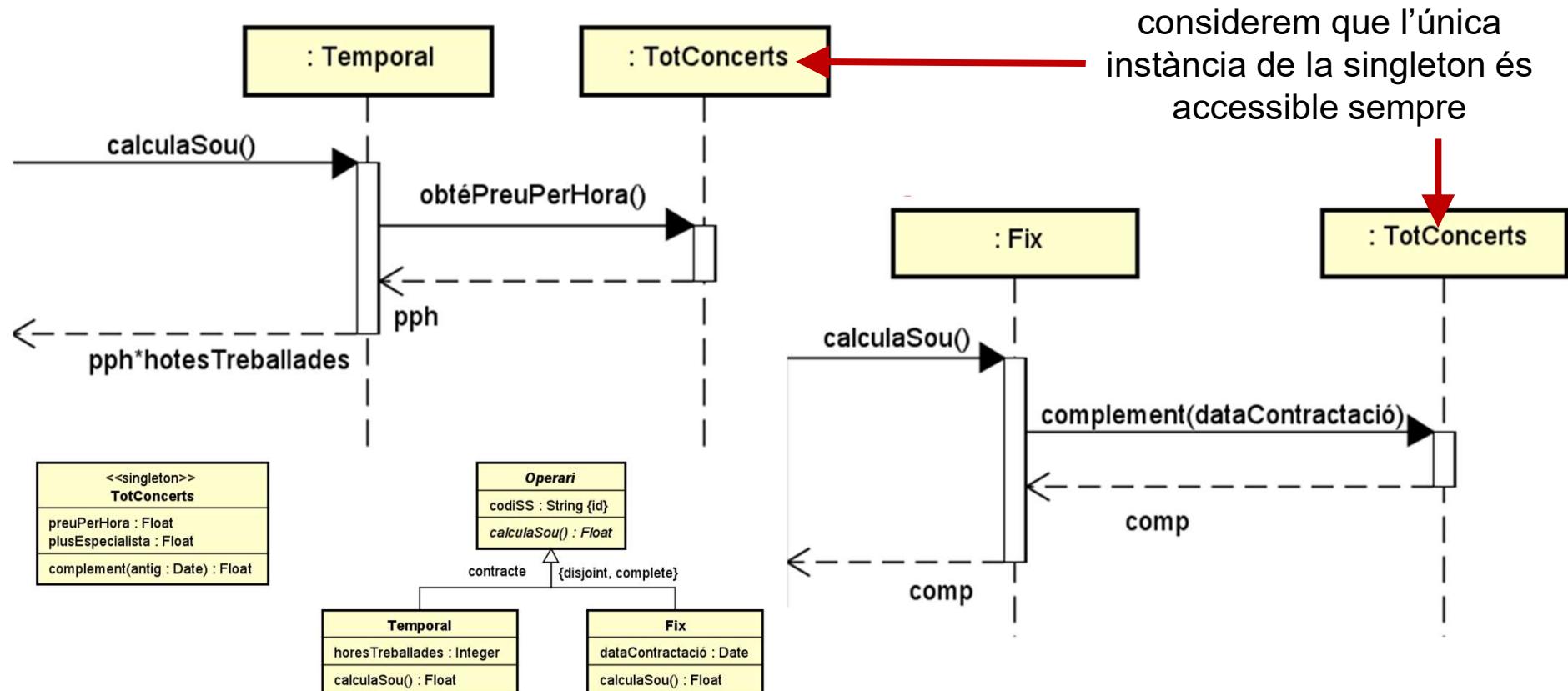
operació Temporal::calculaSou(): Float

retorna el producte de les hores treballades amb el cost per hora

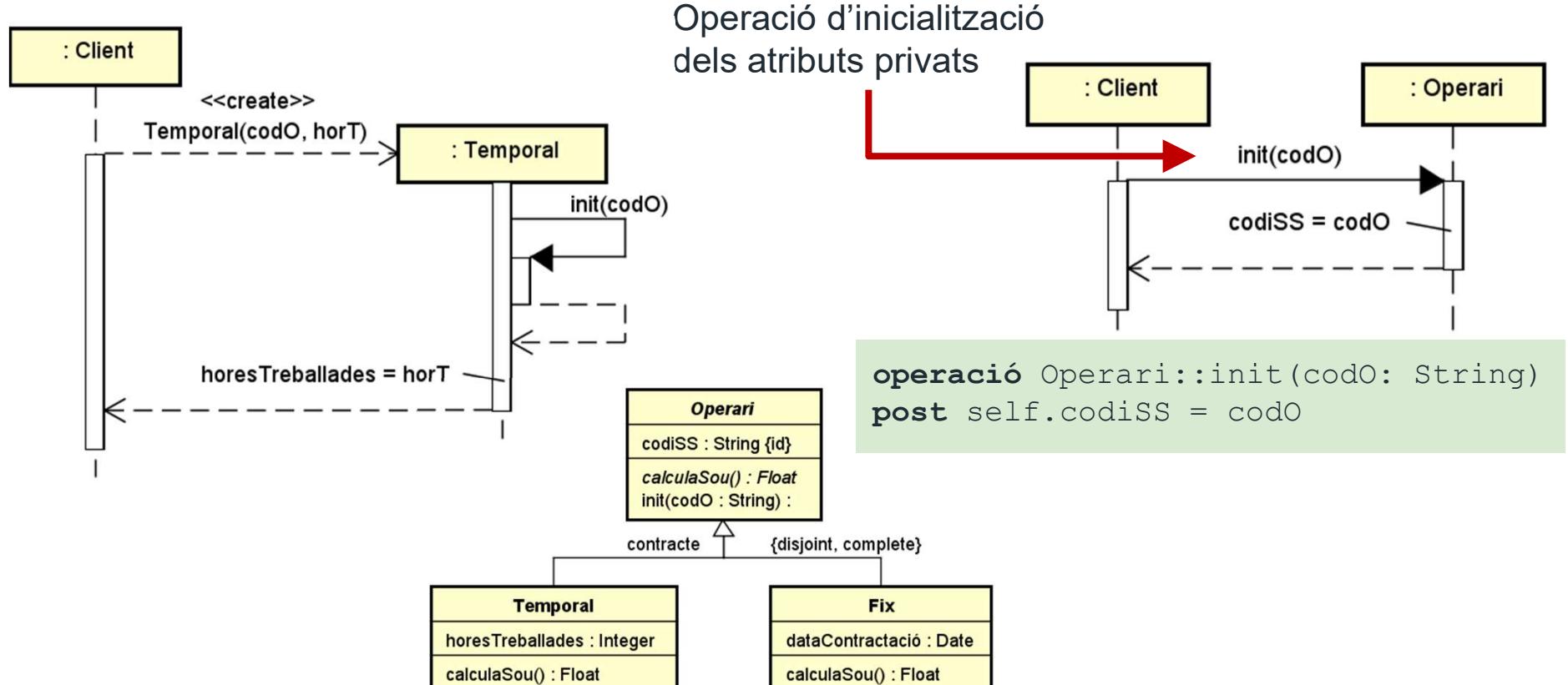
operació Fix::calculaSou(): Float

retorna el sou corresponent a l'antiguitat del contracte

Polimorfisme, exemple: model del comportament (3)



Operacions de creació en jerarquies d'especialització



Polimorfisme i principi Obert – Tancat (OCP)

Principi Obert – Tancat (OCP): les classes haurien de ser:

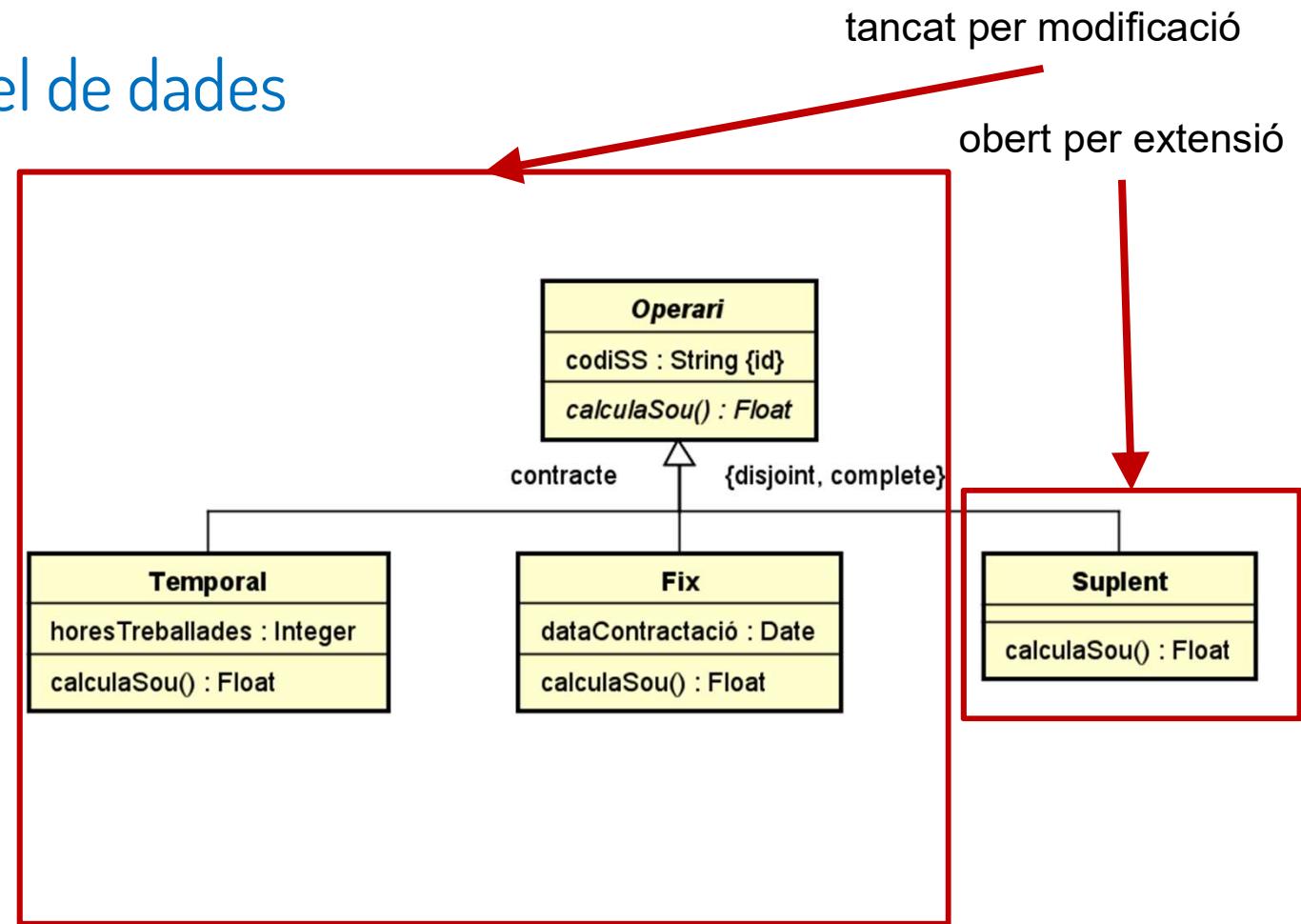
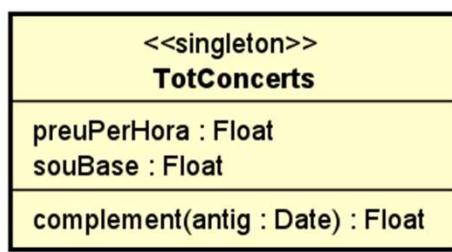
- obertes per a *extensió*: el comportament de la classe es pot estendre per tal de satisfer nous requisits
- tancades per a *modificació*: l'*extensió* no implica canvis en els elements existents de la classe (atributs, operacions i rols)

L'OCP protegeix el sistema envers futures evolucions, afavorint doncs la seva canviabilitat

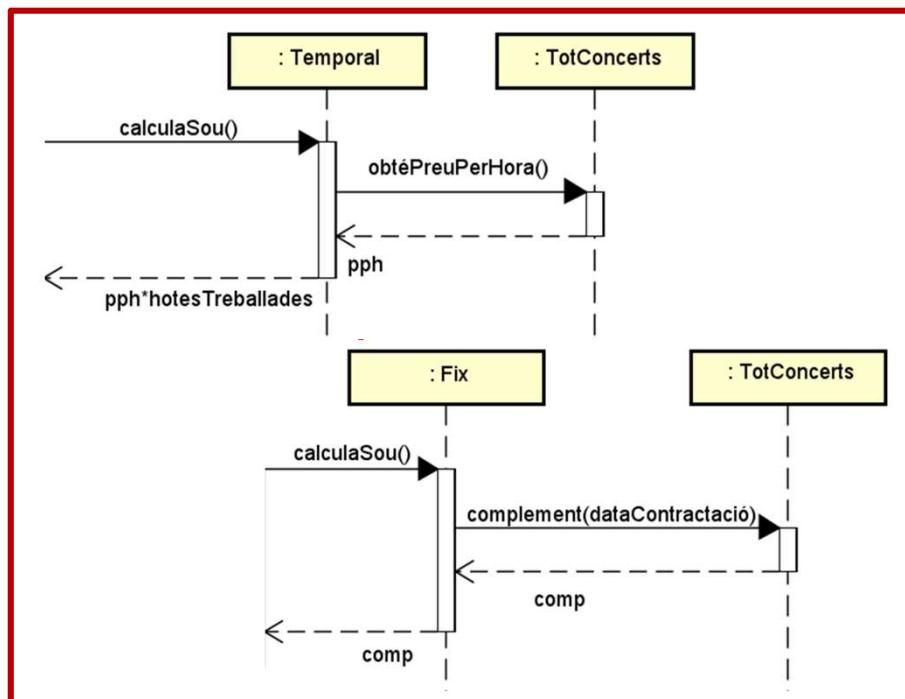
Exemple d'evolució d'un sistema i OCP: enunciat

En base a la seva experiència, l'empresa TotConcerts ha decidit la necessitat de contractar suplents que no cobren més que un sou base

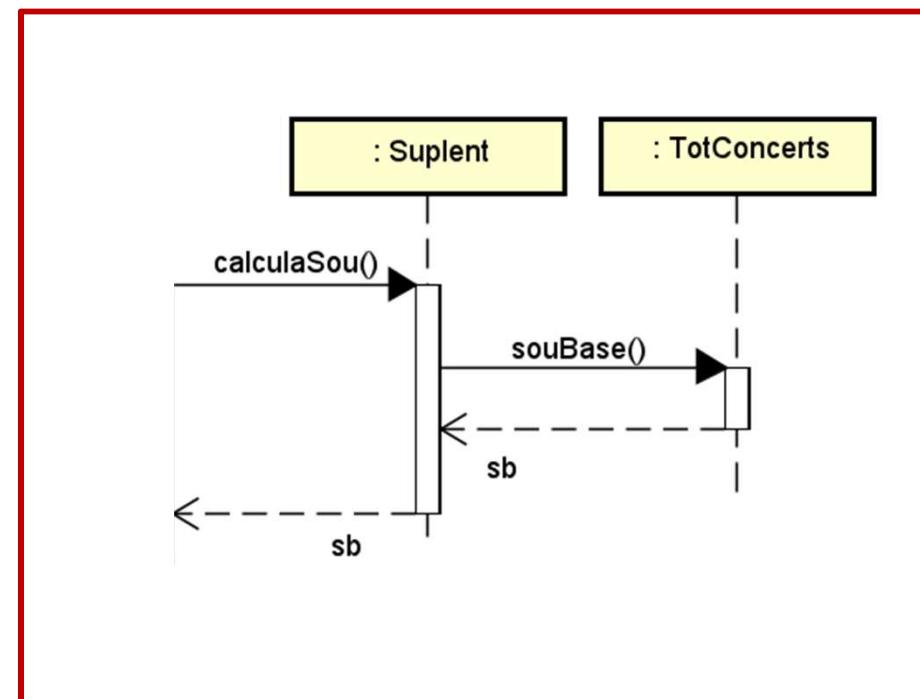
Exemple: model de dades



Exemple: model del comportament



tancat per modificació



obert per extensió

3.3 El procés de disseny

Procés de disseny

La transició de l'especificació al disseny no és trivial

- seguirem un procés sistemàtic
 - combinant els dos patrons arquitectònics elegits
- caldrà integrar en el procés
 - un disseny d'interfície (mínim)
 - el salt conceptual entre els elements d'especificació i els de disseny
 - desapareixen alguns elements (p.e., associacions ternàries i classes associatives, ...) i n'apareixen de nous (p.e., visibilitat i navegabilitat)
- haurem de considerar diverses variants
 - en aquesta assignatura, l'impacte de dos variants en el patró dominant de la capa de domini (*Domain Model* vs. *Transaction Script*)
 - en aquest capítol ens centrem en *Domain Model*, i deixem *Transaction Script* per al capítol 4

Procés de disseny: Punt de partida (Especificació)

Casos d'ús	Diagrama de classes	Diagrames de seq.	Contractes	Disseny interfície
Essencials	Sense limitacions. No hi ha operacions	Un únic objecte per al sistema	De sistema. Amb possibles redundàncies. Només precs.	No existeix

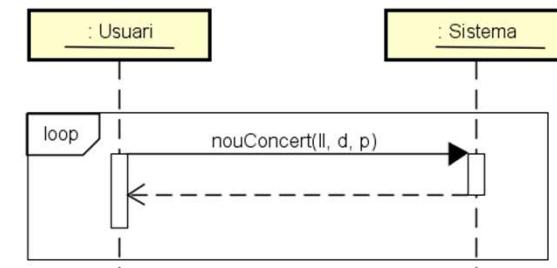
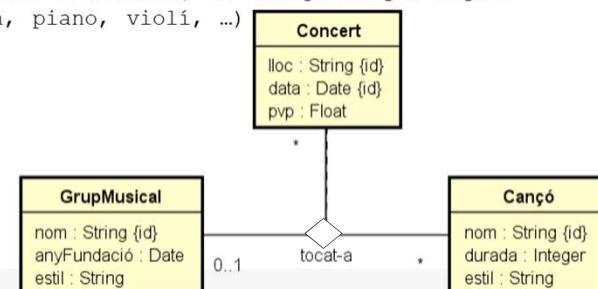
cas d'ús Alta Músic

activació L'Usuari vol donar d'alta un nou músic
escenari principal

1. L'Usuari entra el nom i edat del músic
2. El Sistema enregistra nom i edat
3. L'Usuari entra un a un els noms dels instruments que el músic tocaFinalment, el Sistema mostra el número d'instruments de cada tipus que toca el músic

escenaris alternatius ...

requisits no funcionals En crear el sistema, es van ja afegir alguns instruments bàsics (guitarra, piano, violí, ...)



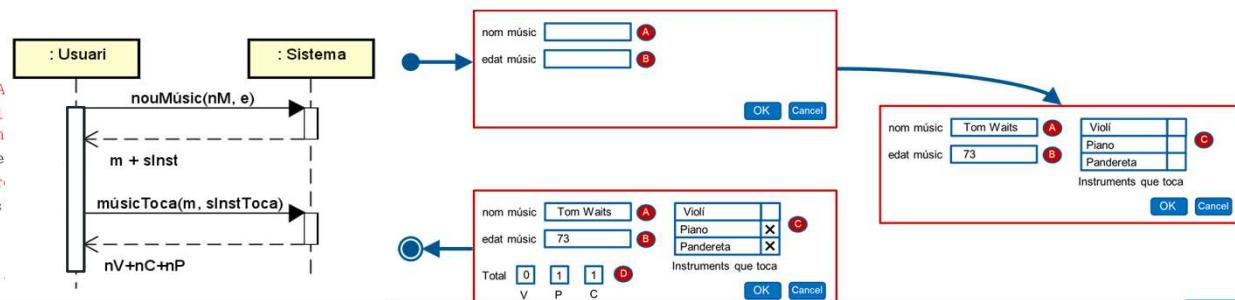
operació nouMúsic (n: String; e: Integer)

post crea una instància de Músic = (n, e, 0)

Procés de disseny: Disseny de la interfície d'usuari

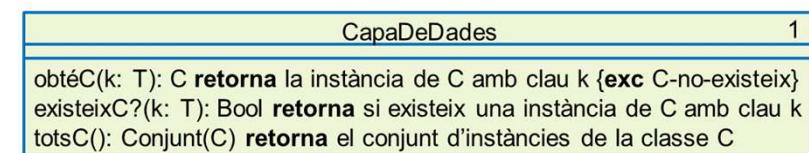
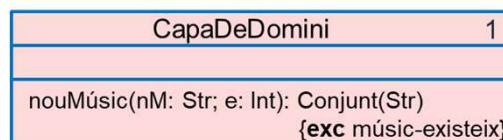
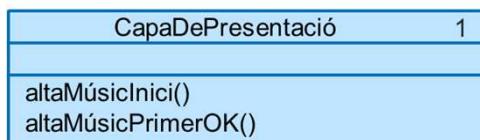
Casos d'ús	Diagrama de classes	Diagrames de seq.	Contractes	Disseny interfície
Essencials	Sense limitacions. No hi ha operacions	Un únic objecte per al sistema	De sistema. Amb possibles redundàncies. Només precs.	No existeix
<u>Concrets</u>	Sense limitacions. No hi ha operacions	1. Disseny de la interfície d'usuari <u>Model del comportament adaptat a la interfície</u>	De sistema. Amb possibles redundàncies. Només precs.	<u>Existeix</u>

cas d'ús Alta Músic
activació L'Usuari vol donar d'alta un nou músic
escenari principal
1. L'Usuari entra el nom i edat del músic en els camp A
2. El Sistema enregistra nom i edat i mostra en la taula registrats en el sistema, juntament amb un camp que indica si el músic toca instruments o no.
3. L'Usuari entra un a un els noms dels instruments que el músic toca i quan acaba, pressiona el botó OK.
4. Finalment, el Sistema mostra el número d'instruments que el músic toca en els camps numèrics corresponents D
escenaris alternatius ...
requisits no funcionals En crear el sistema, es van ja tenir en compte els instruments bàsics (guitarra, piano, violí, ...)



Procés de disseny: Assignació de responsabilitats a capes

Casos d'ús	Diagrama de classes	Diagrames de seq.	Contractes	Disseny interfície
Concrets	Sense limitacions. No hi ha operacions	Model del comportament adaptat a la interfície Un únic objecte per al sistema	De sistema. Amb possibles redundàncies. Només precs.	Existeix
2. Assignació de responsabilitats a capes				
Concrets	Sense limitacions. <u>Operacions de capa</u>	<u>Un objecte per capa</u>	<u>De capes. Auto-continguts.</u> <u>Precs. i excs.</u>	Existeix



La forma concreta de les operacions de la capa de domini i la capa de dades depèn del patró dominant de la capa de domini (*Domain Model vs. Transaction Script*)

Procés de disseny: Simplificació del model

Casos d'ús	Diagrama de classes	Diagrames de seq.	Contractes	Disseny interfície
Concrets	Sense limitacions. Operacions de capa	Un objecte per capa	De capes. Auto-continguts. Precs. i excs.	Existeix
Concrets	<u>Sense elements no suportats.</u> Operacions de capa	Un objecte per capa	<u>De capes (en la capa de domini, adaptats).</u> Auto-continguts. Precs. i excs.	Existeix

3. Simplificació del model

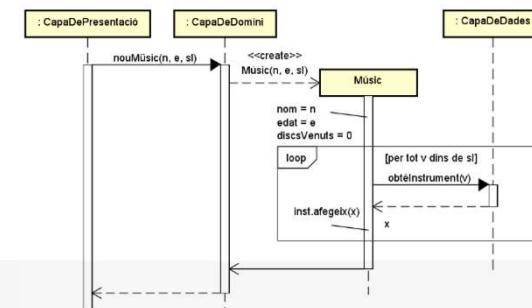
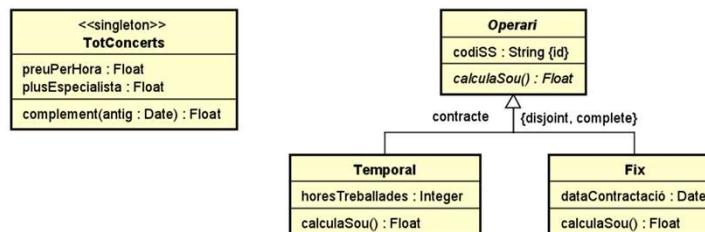


Les regles concretes que regeixen el procés de simplificació del model depenen del patró dominant de la capa de domini (*Domain Model vs. Transaction Script*)

Procés de disseny: Disseny de les operacions

Casos d'ús	Diagrama de classes	Diagrames de seq.	Contractes	Disseny interfície
Concrets	Sense elements no suportats. Operacions de capa	Un objecte per capa	De capes (en la capa de domini, adaptats). Auto-continguts. Precs. i excs.	Existeix
Concrets	Sense elements no suportats. <u>Operacions de classe</u>	<u>Tants objectes com calgui</u>	<u>De classe, incloent-ne operacions auxiliars.</u> Auto-continguts. Precs. i excs.	Existeix

4. Disseny de les operacions



Procés de disseny: Resum

Casos d'ús	Diagrama de classes	Diagrames de seq.	Contractes	Disseny interfície
Essencials	Sense limitacions. No hi ha operacions	Un únic objecte per al sistema	De sistema. Amb possibles redundàncies. Només precs.	No existeix
1. Disseny de la interfície d'usuari				
Concrets	Sense limitacions. No hi ha operacions	Un únic objecte per al sistema	De sistema. Amb possibles redundàncies. Només precs.	<u>Existeix</u>
2. Assignació de responsabilitats a capes				
Concrets	Sense limitacions. <u>Operacions de capa</u>	<u>Un objecte per capa</u>	<u>De capes. Auto-continguts.</u> <u>Precs. i excs.</u>	Existeix
3. Simplificació del model				
Concrets	<u>Sense elements no suportats.</u> Operacions de capa	Un objecte per capa	De capes (<u>en la capa de domini, adaptats</u>). Auto-continguts. Precs. i excs.	Existeix
4. Disseny de les operacions				
Concrets	Sense elements no suportats. <u>Operacions de classe</u>	<u>Tants objectes com calgui</u>	<u>De classe, incloent-ne operacions auxiliars.</u> Auto-continguts. Precs. i excs.	Existeix

3.4 Disseny de la interfície d'usuari

Casos d'ús	Diagrama de classes	Diagrams de seq.	Contractes	Disseny interfície
Essencials	Sense limitacions. No hi ha operacions	Un únic objecte per al sistema	De sistema. Amb possibles redundàncies. Només precs.	No existeix
1. Disseny de la interfície d'usuari				
Concrets	Sense limitacions. No hi ha operacions	Un únic objecte per al sistema	De sistema. Amb possibles redundàncies. Només precs.	<u>Existeix</u>

Context

Els casos d'ús descriuen les funcionalitats del sistema de forma abstracta

- el model del comportament no considera la interfície d'usuari

Però a l'hora de fer el disseny, cal tenir en compte la comunicació de l'usuari i el sistema:

- el disseny de la interfície d'usuari impacta en les interaccions (**esdeveniments de presentació**) i el flux de dades entre el usuari i el sistema
- necessitem doncs adaptar el model del comportament a la interfície abans de començar el disseny pròpiament dit

Disseny de la interfície

Cal identificar els elements *relevants* de la interfície que permeten la comunicació entre els usuaris i el sistema

- disseny de la interfície amb camps individuals, zones de missatge, ...
- no ens preocupen de temes estètics (colors, zooms, scroll, etc.)
- identifiquen els elements per referir-nos-hi

The screenshot shows a user interface for inputting information about a musician. It includes fields for 'nom músic' (name) containing 'Tom Waits' (labeled A), 'edat músic' (age) containing '73' (labeled B), and a 'Total' section with three buttons labeled 0, 1, and 1 (labeled D). Below these are buttons V, P, and C. To the right is a list of instruments ('Instruments que toca') with 'Violí' checked (labeled C), while 'Piano' and 'Pandereta' are unchecked. At the bottom are 'OK' and 'Cancel' buttons.

nom músic	Tom Waits	A		
edat músic	73	B		
Total	0	1	1	D
	V	P	C	
Instruments que toca				
Violí				
Piano			X	
Pandereta			X	

OK Cancel

Casos d'ús individuals concrets (1)

Incorporen la comunicació amb l'usuari a l'especificació

cas d'ús Alta Músic

activació L'Usuari vol donar d'alta un nou músic
escenari principal

1. L'Usuari entra el nom i edat del músic
2. El Sistema enregistra nom i edat

3. L'Usuari entra un a un els noms dels instruments que el músic toca

4. Finalment, el Sistema mostra el número d'instruments de cada tipus que toca el músic

escenaris alternatius ...

requisits no funcionals En crear el sistema, es van ja afegir alguns instruments bàsics (guitarra, piano, violí, ...)

Casos d'ús individuals concrets (2)

Incorporen la comunicació amb l'usuari a l'especificació

cas d'ús Alta Músic

activació L'Usuari vol donar d'alta un nou músic
escenari principal

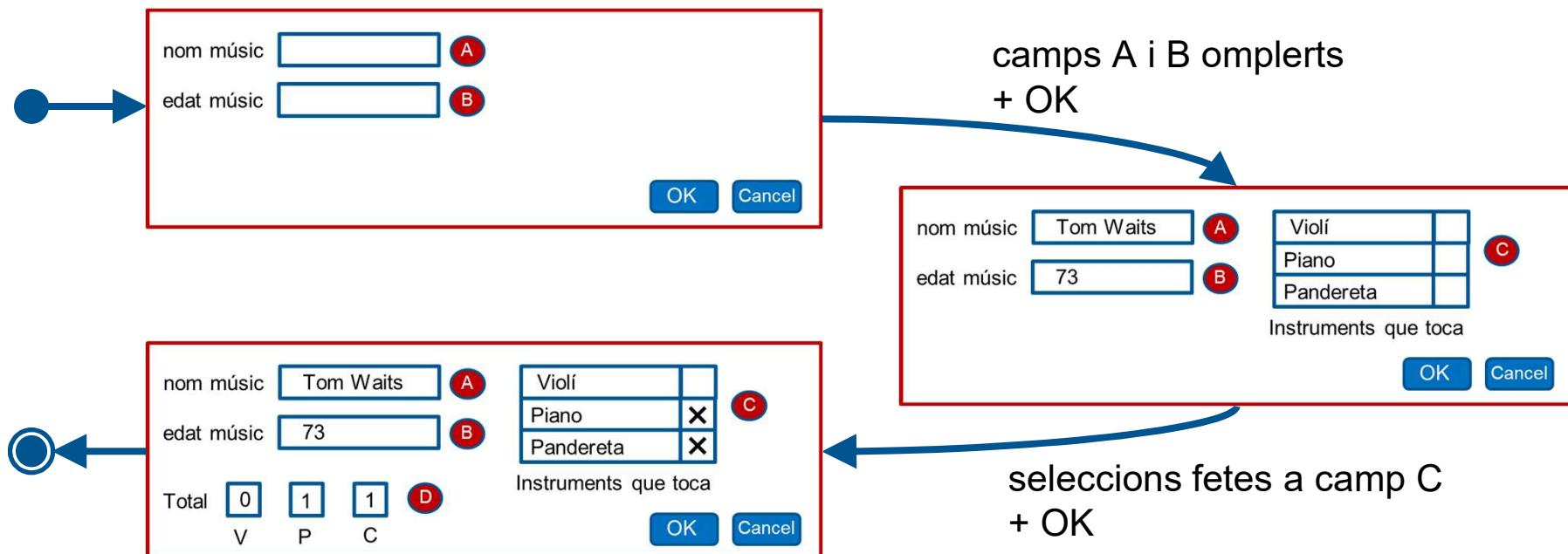
1. L'Usuari entra el nom i edat del músic **en els camp A i B** i prem OK
2. El Sistema enregistra nom i edat **i mostra en la taula C tots els instruments registrats en el sistema**, juntament amb un camp individual de selecció
3. **L'Usuari entra un a un els noms dels instruments que el músic toca**
L'Usuari selecciona els instruments i quan acaba, prem OK
4. Finalment, el Sistema mostra el número d'instruments de cada tipus que toca el músic **en els camps numèrics corresponents D**

escenaris alternatius ...

requisits no funcionals En crear el sistema, es van ja afegir alguns instruments bàsics (guitarra, piano, violí, ...)

Mapa navegacional

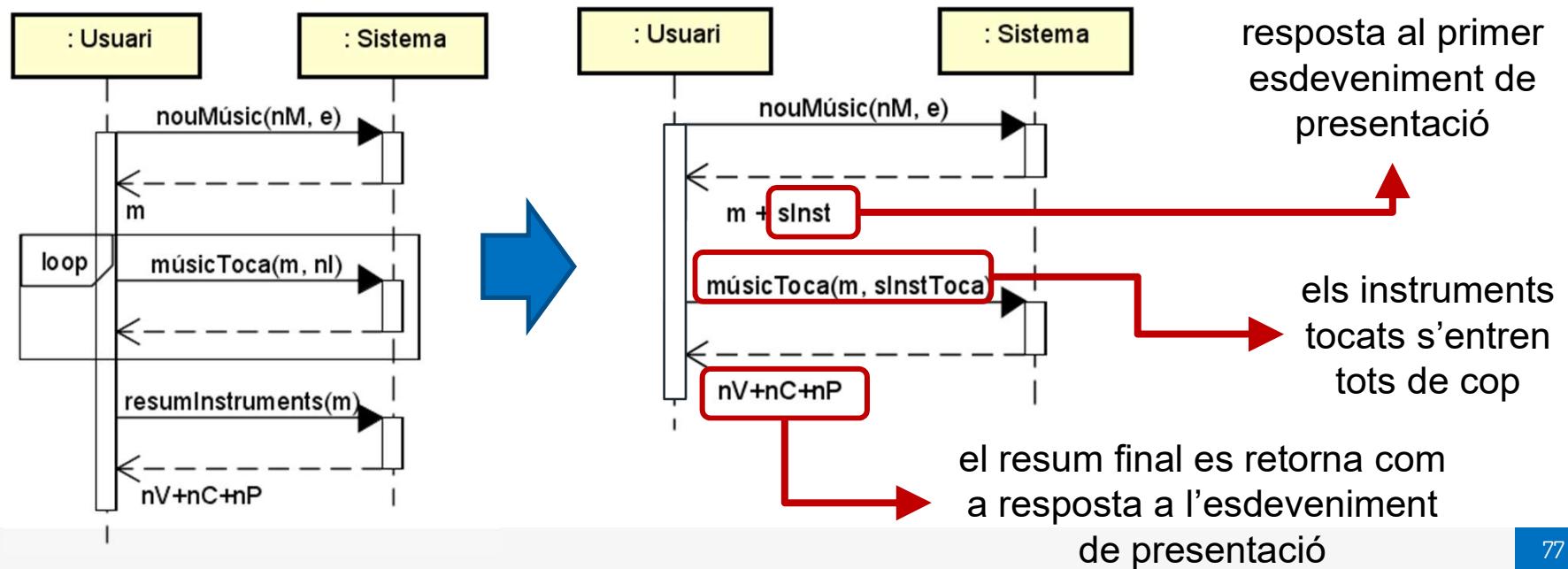
Resumeix l'evolució de la interfície al llarg del cas d'ús i mostra clarament els esdeveniments de presentació



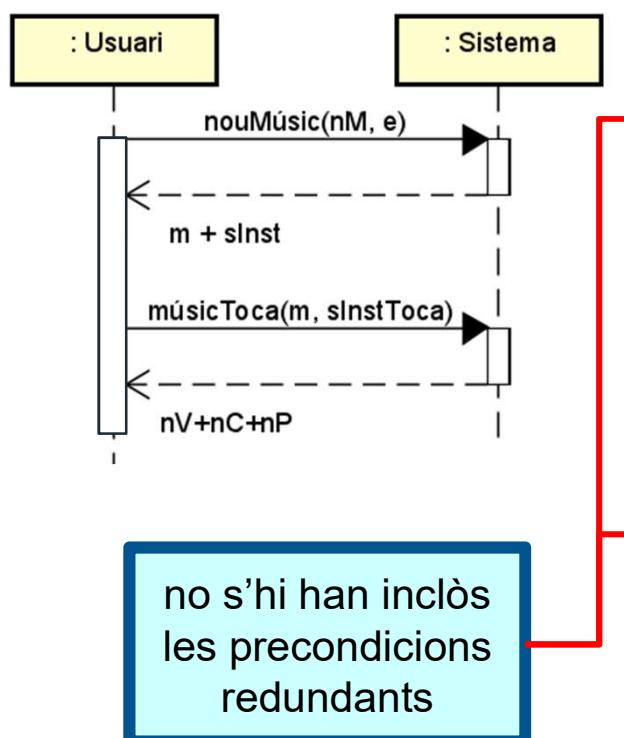
Impacte en el model del comportament (1)

La consideració de la interfície d'usuari pot exigir

- canvis en el diagrama de seqüència, i/o
- canvis en els contractes



Impacte en el model del comportament (2)



operació nouMúsic (nM: String; e: Integer):
Músic + Conjunt(String)
post crea una instància m de Músic = (nM, e, 0)
retorna m + conjunt dels noms de tots els Instruments guardats al sistema

operació músicToca (m: Músic;
sinstToca: Conjunt(String)):
Integer+Integer+Integer
pre per tot x dins de sInstToca: existeix al sistema un Instrument x amb nom nI
post per tot x dins de sInstToca: s'enregistra que m toca x
retorna el total d'instruments de cada tipus que toca el músic, respectivament: vent, corda i percussió

3.5 Assignació de responsabilitats a capes

Casos d'ús	Diagrama de classes	Diagrams de seq.	Contractes	Disseny interfície
<u>Concrets</u>	Sense limitacions. No hi ha operacions	<u>Model del comportament adaptat a la interfície</u> Un únic objecte per al sistema	De sistema. Amb possibles redundàncies. Només precs.	<u>Existeix</u>
2. Assignació de responsabilitats a capes				
Concrets	Sense limitacions. <u>Operacions de capa</u>	<u>Un objecte per capa</u>	<u>De capes. Auto-continguts.</u> <u>Precs. i excs.</u>	Existeix

Context

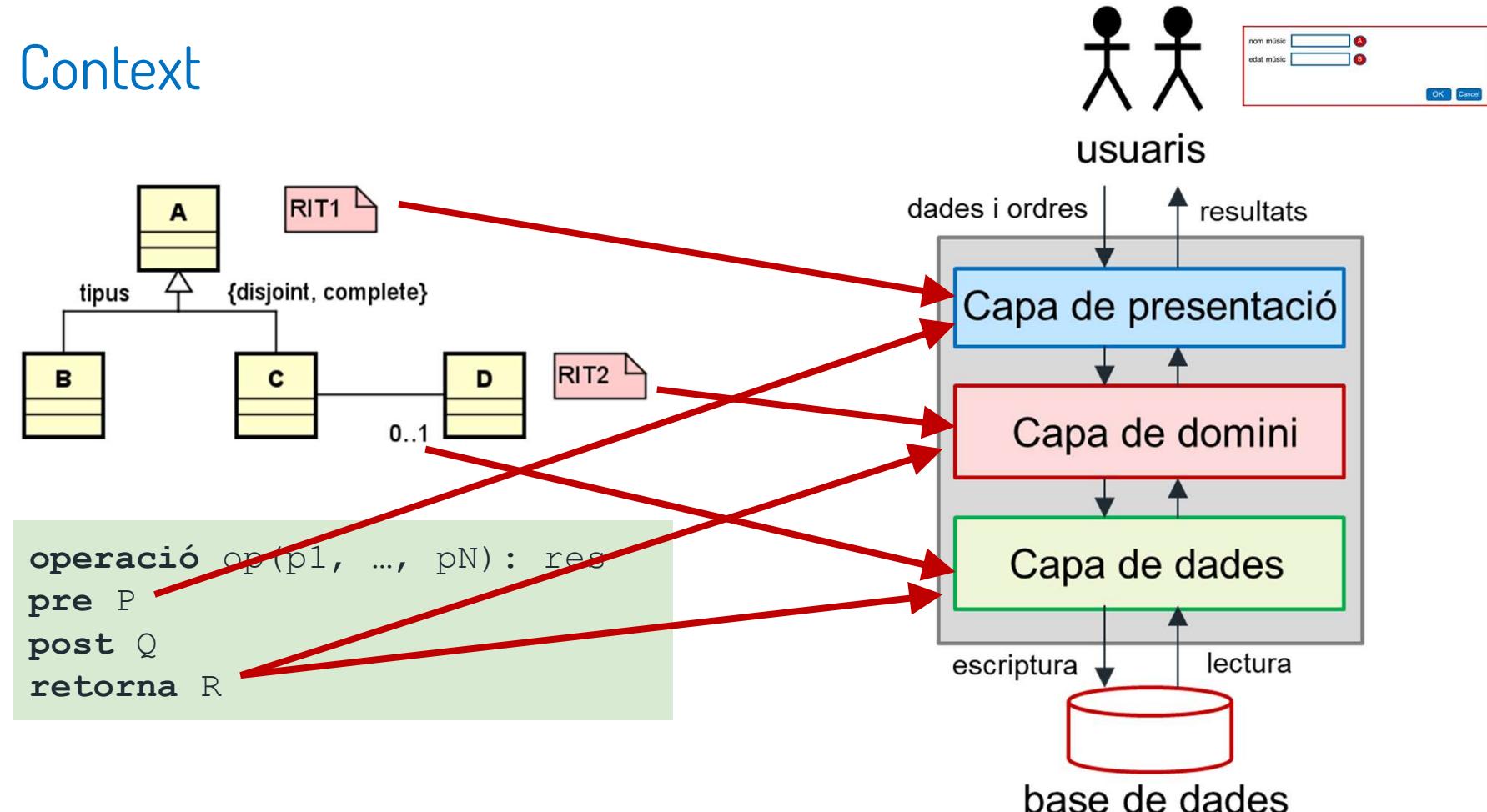
Partim d'una interfície d'usuari i d'un model que conté:

- l'estructura de les dades que el sistema ha de gestionar
- les operacions que el sistema ha d'implementar (adaptades a la interfície d'usuari ja dissenyada)

En el disseny usant el patró arquitectònic en tres capes, cal decidir quines **responsabilitats** s'assignen a cada capa:

- quina capa manté la consistència de les dades
- quina capa comprova les precondicions de les operacions
- quina capa executa les postcondicions de les operacions
- quina capa genera els resultats de les operacions

Context

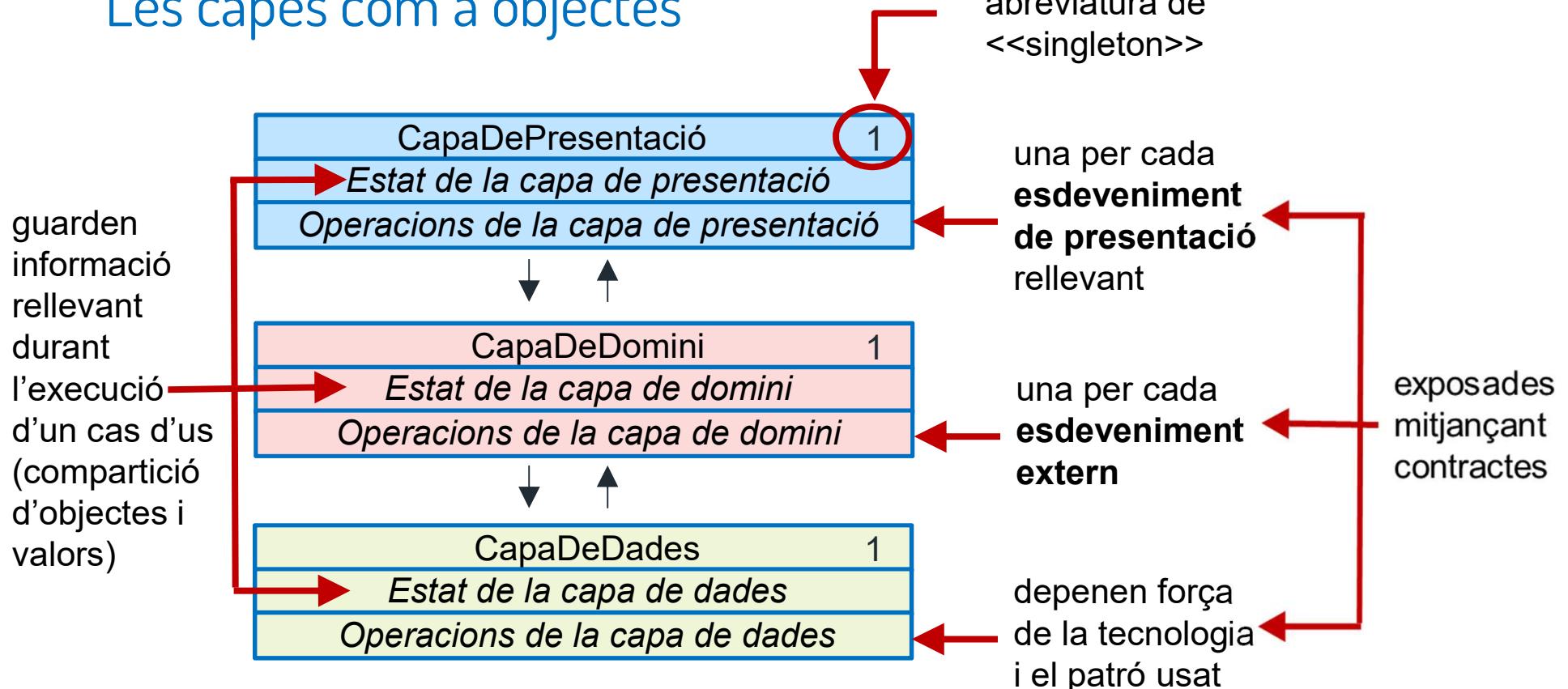


Les capes com a objectes

En una primera aproximació, representem cada capa com a una classe amb un únic objecte (*singleton*)

- (spoiler) Aquestes classe s'anomenen **controladors façana**
 - són un primer exemple (conceptualment simple) de patró de disseny
 - veurem més endavant altres alternatives
 - però de moment, ja ens va bé aquesta opció!
- els controladors façana defineixen totes les operacions necessàries a la capa i les ofereixen als seus clients mitjançant un **contracte**
 - concepte de **disseny per contracte**

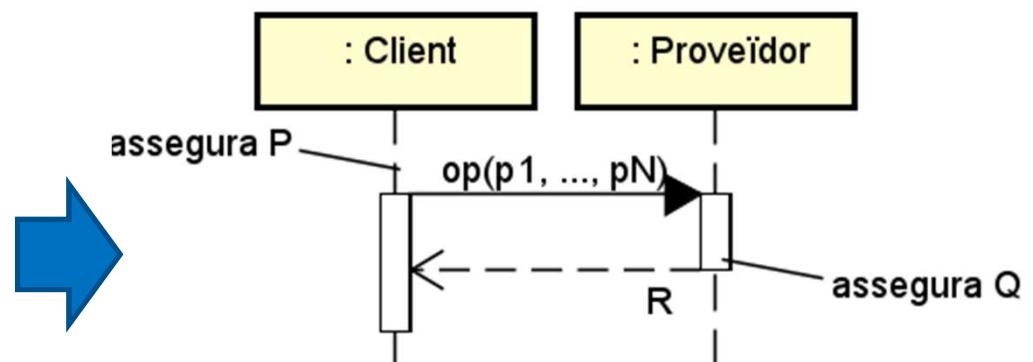
Les capes com a objectes



Disseny per contracte

Aproximació clàssica a la programació, adoptada al disseny,
guiada per les operacions

```
operació op(p1, ..., pN) : res
  pre P
  post Q
  retorna R
```



Si la invocació d' op assegura que es compleix P , aleshores op ha d'assegurar que l'estat final del sistema compleix Q i que retorna el valor R

Tractament dels errors en el model del comportament

Problema: estem obligant a que les precondicions les comprovi sempre el client que crida l'operació:

- pot ser feixuc
- fins i tot, pot ser impossible

A disseny, distingim en les operacions un segon tipus de comprovació d'error: les **excepcions**

- la feina de comprovar l'excepció passa a ser responsabilitat del proveïdor de l'operació, i no del seu client

Les excepcions en el model

```
operació op(p1, ..., pN): res
pre P
pre E
post Q
retorna R
```

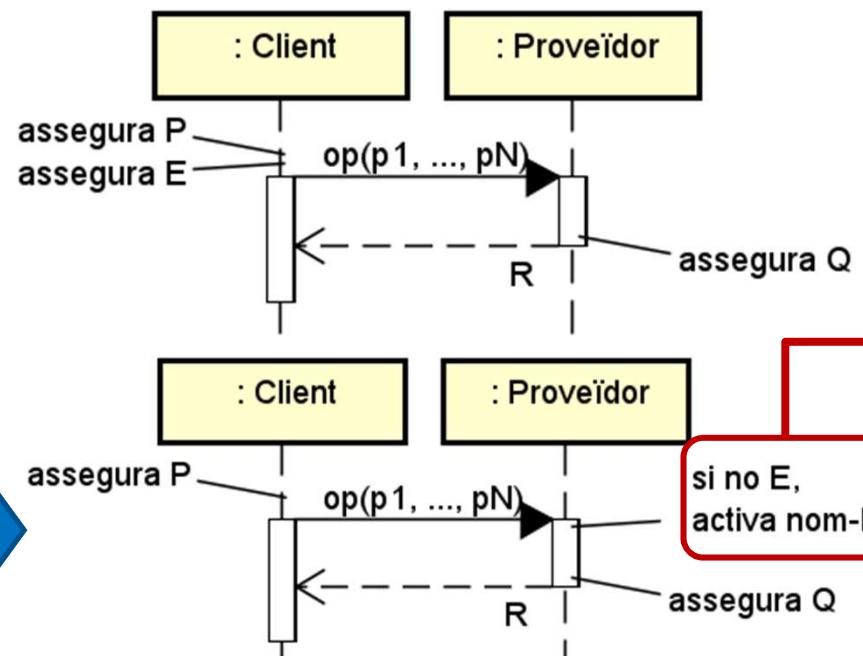
```
operació op(p1, ..., pN): res
pre P
exc nom-E: no E
post Q
retorna R
```

nova clàusula
en el contracte

cal negar la
precondició

les excepcions tenen nom

l'operació **activa** l'excepció si
detecta la condició



Convenció: quan s'activa una excepció,
es desfan automàticament els possibles
canvis d'estat fets pel cas d'ús

Exemple

Suposem que el procés de disseny ens ha portat a l'operació següent definida a la capa de domini:

```
operació CapaDeDomini::nouMúsic (n: String; e: Integer)
pre e ≥ 0
pre no existeix al sistema cap Músic amb nom n
post crea una instància de Músic = (n, e, 0)
```

Si considerem el context en què aquesta operació serà invocada per la capa de presentació (v. Secció 3.2), com queden les dues precondicions?

Exemple

- $e \geq 0$: fàcilment comprovable per la capa de presentació
- no existeix al sistema cap Músic amb nom n: la capa de presentació no té cap mitjà per comprovar aquesta condició



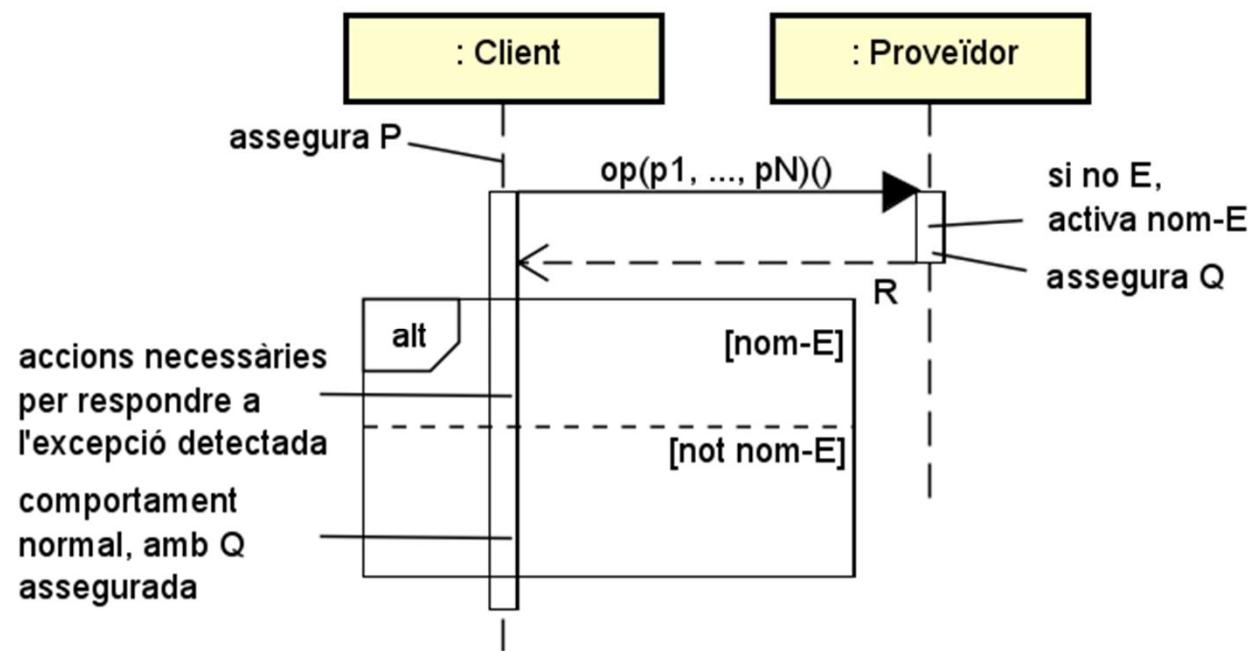
```
operació CapaDeDomini::nouMúsic (n: String; e: Integer)
pre e ≥ 0
exc músic-existeix: existeix al sistema un Músic amb nom n
post crea una instància de Músic = (n, e, 0)
```

Tractament de les excepcions

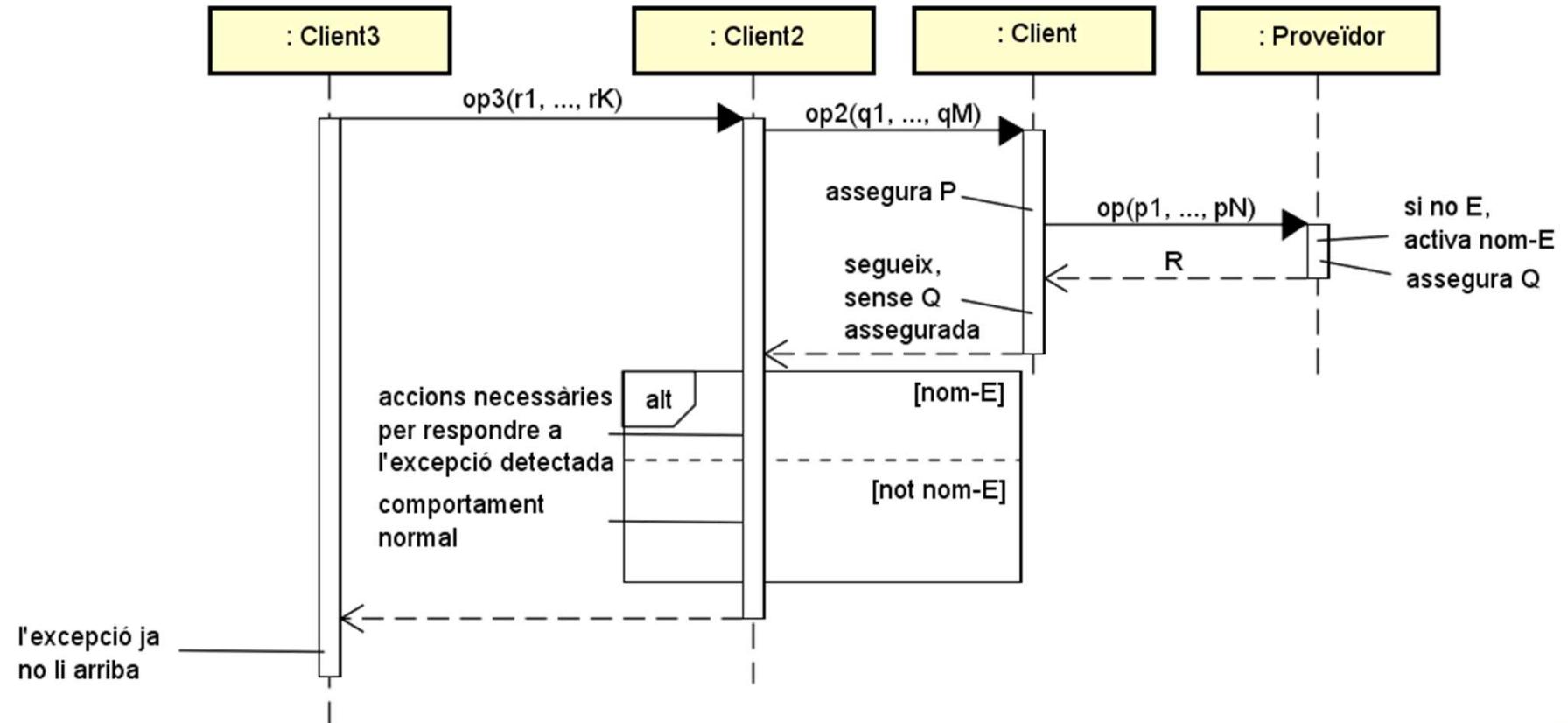
Quan una operació activa una excepció, el client pot:

- **capturar** (i.e., tractar) l'excepció
- **propagar** l'excepció (no fent res)
 - simplement passa al client del client, i així successivament
 - en algun moment, l'excepció s'hauria de capturar
 - a no ser que es consideri irrelevat (perillós...)
 - en la propagació, també es van desfent els canvis dels clients trobats a la cadena de crides

Captura d'excepcions



Propagació d'excepcions



Eliminació de redundància

Les precondicions i excepcions asseguraran que es compleixen les restriccions derivades del model de les dades

- restriccions gràfiques
 - multiplicitats, qualificadors especialització, i les pròpies del diagrama
- restriccions textuais

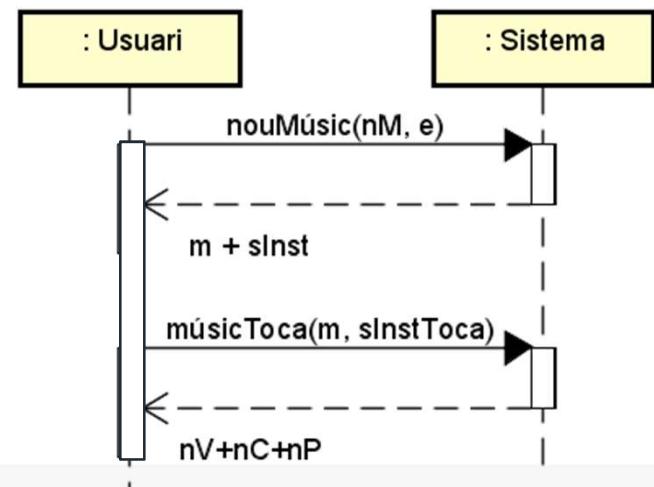
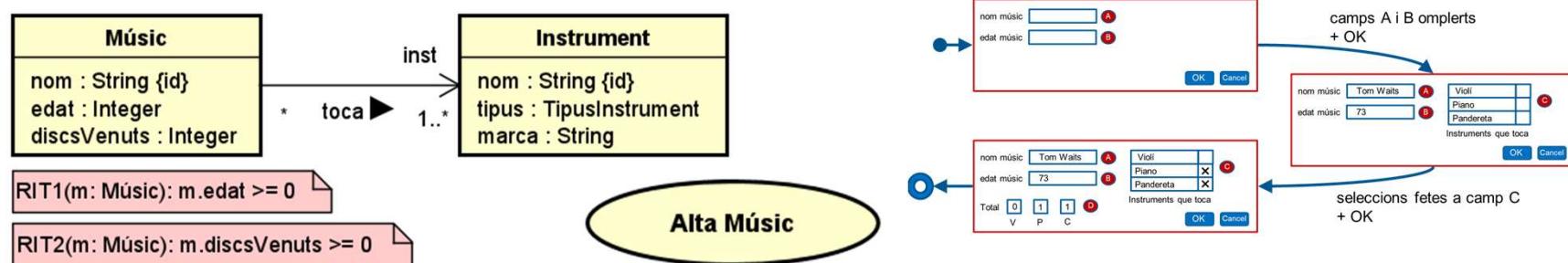
Granularitat de cas d'ús: les restriccions sobre les dades:

- es compleixen abans i després d'executar el cas d'ús
- es poden violar temporalment durant l'execució del cas d'ús
- es poden comprovar en el moment més adient

Responsabilitats típiques de les capes – capa de presentació

- ofereix una operació per cada esdeveniment de presentació de cada cas d'ús concret
 - entre elles, una d'iniciació del cas d'ús
- s'encarrega d'aquelles precondicions purament sintàctiques
 - p.e., comprovar que un número està en un rang de valors
- assegura, atès el disseny de la interfície d'usuari, que altres precondicions no es violaran mai
 - p.e., assegura que se selecciona algun element d'un desplegable
- delega en la capa de domini aquelles responsabilitats que no pot solucionar per si sola
 - eventualment, pre-processant algunes dades per facilitar-li la feina
- processa si cal, i mostra, els resultats produïts per la capa de domini

Exemple – punt de partida (després de disseny interfície)



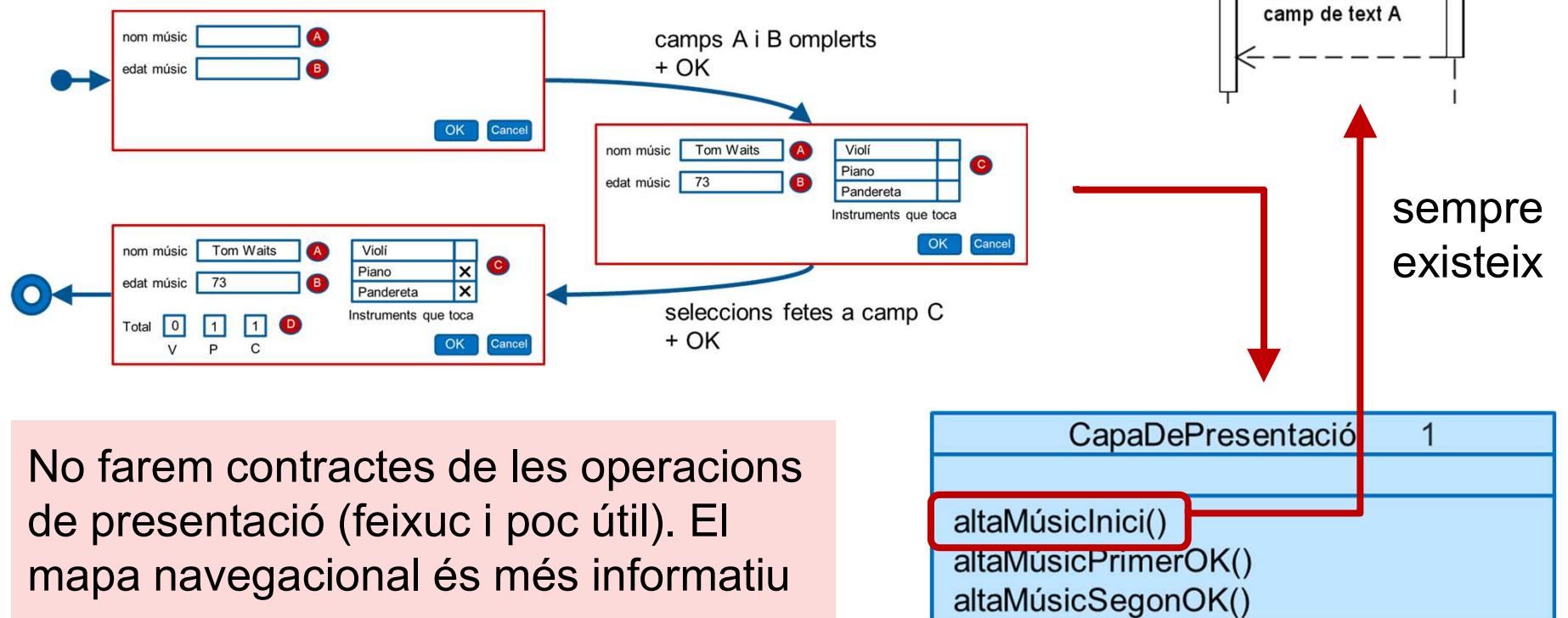
```

operació nouMúsic (nM: String; e: Integer):
    Músic + Conjunt(String)
post crea una instància m de Músic = (nM, e, 0)
retorna m + conjunt dels noms de tots els
    Instruments guardats al sistema
  
```

```

operació músicToca (m: Músic;
    sInstToca: Conjunt(String)):
    Integer+Integer+Integer
pre per tot x dins de sInstToca: existeix al
    sistema un Instrument x amb nom nI
post per tot x dins de sInstToca: s'enregistra
    que m toca x
retorna el total d'instruments de cada tipus que
    toca el músic, respectivament: vent,
    corda i percussió
  
```

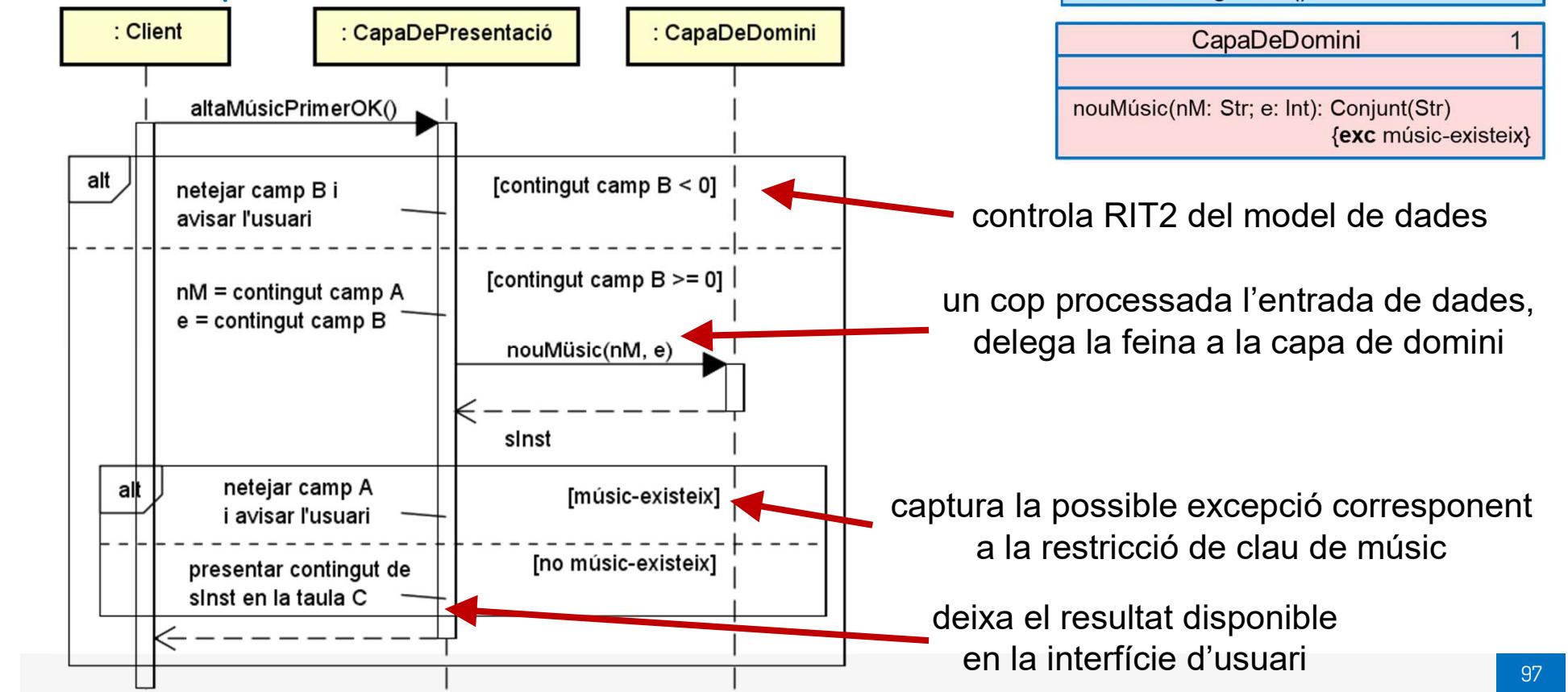
Exemple – capa de presentació, responsabilitats



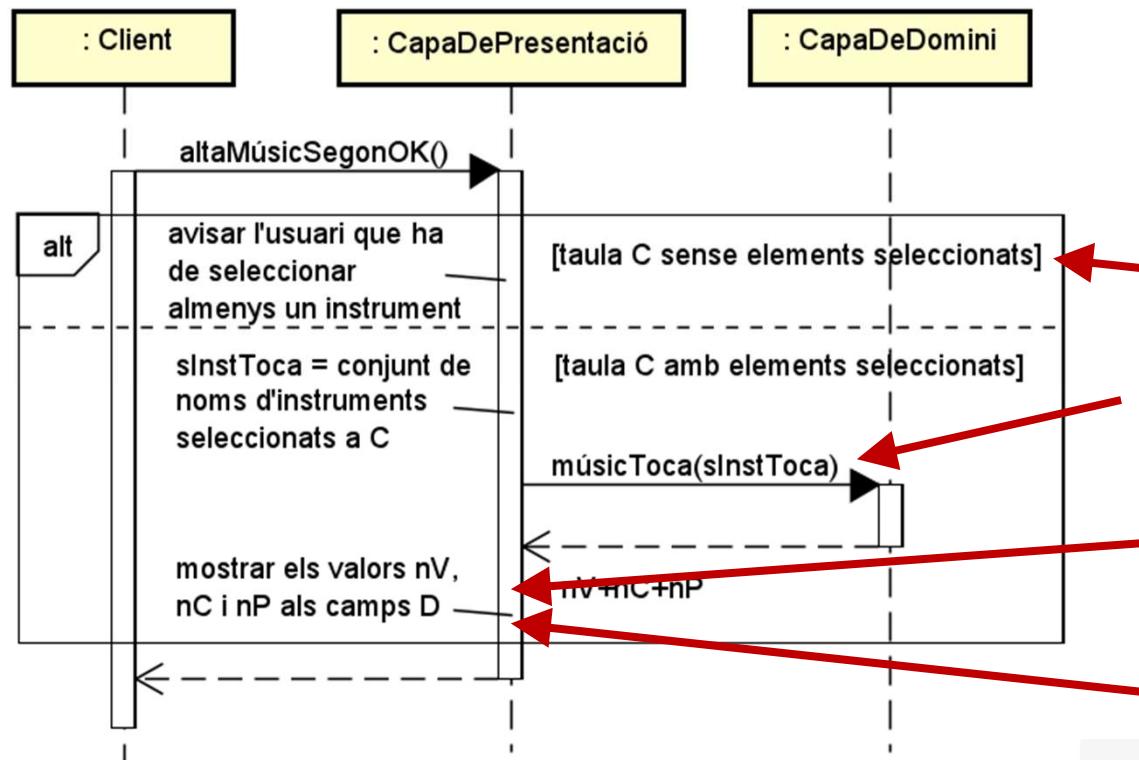
Responsabilitats típiques de les capes – capa de domini

- implementa la lògica de negoci del problema
 - per això, les seves operacions són bàsicament les que s'han identificat a l'especificació, ajustades als requisits de les altres capes
 - sobretot, a partir del disseny de la capa de presentació
 - degut a la seva complexitat, farà ús intensiu de patrons de disseny orientats a objectes
- aplica disseny per contracte per gestionar els errors
 - les precondicions d'especificació tractades per la capa de presentació, romanen precondicions a la capa de domini
 - excepcionalment, es poden definir com a excepcions per reusabilitat
 - les precondicions d'especificació no tractades per la capa de presentació, esdevenen excepcions a la capa de domini

Exemple - de la capa de presentació a la capa de domini



Exemple - de la capa de presentació a la capa de domini



CapaDePresentació	1
altaMúsicInici()	
altaMúsicPrimerOK()	
altaMúsicSegonOK()	
CapaDeDomini	1
nouMúsic(nM: Str; e: Int): Conjunt(Str) {exc músic-existeix}	
músicToca(nM: Str; sInstToca: Conjunt(Str)): Int+Int+Int	

controla 1..* del model de dades

un cop processada l'entrada de dades, delega la feina a la capa de domini

no hi ha excepcions a controlar

deixa el resultat disponible en la interfície d'usuari

Exemple – capa de domini, responsabilitats

comprometen reusabilitat
en altres casos d'ús

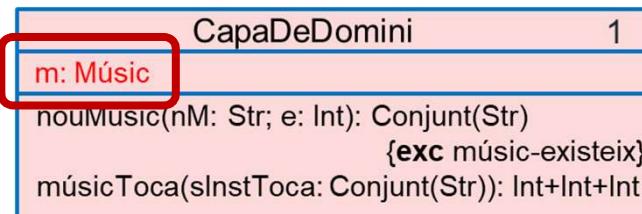
CapaDeDomini	1
nouMúsic(nM: Str; e: Int): Conjunt(Str) {exc músic-existeix} músicToca(nM: Str; sInstToca: Conjunt(Str)): Int+Int+Int	

operació CapaDeDomini::nouMúsic (nM: String; e: Integer): Conjunt (String)
pre e >= 0
exc músic-existeix: já existeix un músic de nom nM
post crea una instància m de Músic = (nM, e, 0)
retorna conjunt dels noms de tots els Instruments guardats al sistema

operació CapaDeDomini::músicToca (nM: String; sInstToca: Conjunt (String)) :
Integer + Integer + Integer
pre per tot x dins de sInstToca: existeix un instrument amb nom x
pre el músic m de nom nomM no toca cap instrument
post per tot x dins de sInstToca: s'enregistra que m toca x
retorna el total d'instruments de cada tipus que toca m: vent, corda i percussió

Exemple – capa de domini, responsabilitats, amb estat

compromet
reusabilitat en
altres casos d'ús



cal modificar el diagrama
de seqüència

```
operació CapaDeDomini::nouMúsic (nM: String; e: Integer): Conjunt(String)
pre e >= 0
exc músic-existeix: já existeix un músic de nom nM
post crea una instància m de Músic = (nM, e, 0)
post self.m = m
retorna conjunt dels noms de tots els Instruments guardats al sistema
```

```
operació CapaDeDomini::músicToca (nM: String; sInstToca: Conjunt(String)) :
    Integer + Integer + Integer
pre per tot x dins de sInstToca: existeix un instrument amb nom x
pre el músic m de nom nomM no toca cap instrument
post per tot x dins de sInstToca: s'enregistra que m toca x
retorna el total d'instruments de cada tipus que toca m: vent, corda i percussió
```

Responsabilitats típiques de les capes – capa de dades

- fa d'intermediària entre la capa de domini i la base de dades
 - recolza doncs la independència tecnològica del sistema...
 - ...afavorint principis de qualitat (canviabilitat, portabilitat, usabilitat)
- assegura que les dades del sistema són persistents
- proporciona operacions de consulta per accedir a les dades del sistema
 - p.e., accés per clau, obtenció de tots els elements
- els detalls de funcionament depenen completament del patró dominant de la capa de domini elegit (v. transparència següent):
 - *Domain Model vs. Transaction Script*
- eventualment, pot aprofitar funcionalitat del sistema gestor de la base de dades per facilitar el disseny i afavorir l'eficiència
 - p.e., ús de disparadors (*triggers*) per activar excepcions

Comunicació capa de domini – capa de dades

La comunicació entre les capes de domini i de dades ha de cobrir la distància conceptual entre dos paradigmes diferents:

- el model O.O. de l'especificació
- la tecnologia relacional de la base de dades

Existeixen dues grans estratègies que cristal·litzen en dos **patrons de disseny** dominants en la capa de domini:

- patró **Domain Model**: domina el model O.O.
 - combina amb el patró **Data Mapper** de la capa de dades
- patró **Transaction Script**: domina la tecnologia relacional
 - combina amb el patró **Row Data Gateway** de la capa de dades
 - els veurem al capítol 4

Patró *Domain Model* (capa de domini)

La capa de domini respon a la visió clàssica O.O.:

- objectes que col·laboren en resposta a esdeveniments externs
- considera la capa de dades com una classe que té un únic objecte
- ús intensiu de patrons de disseny de granularitat més fina

Patró *Data Mapper*(capa de dades)

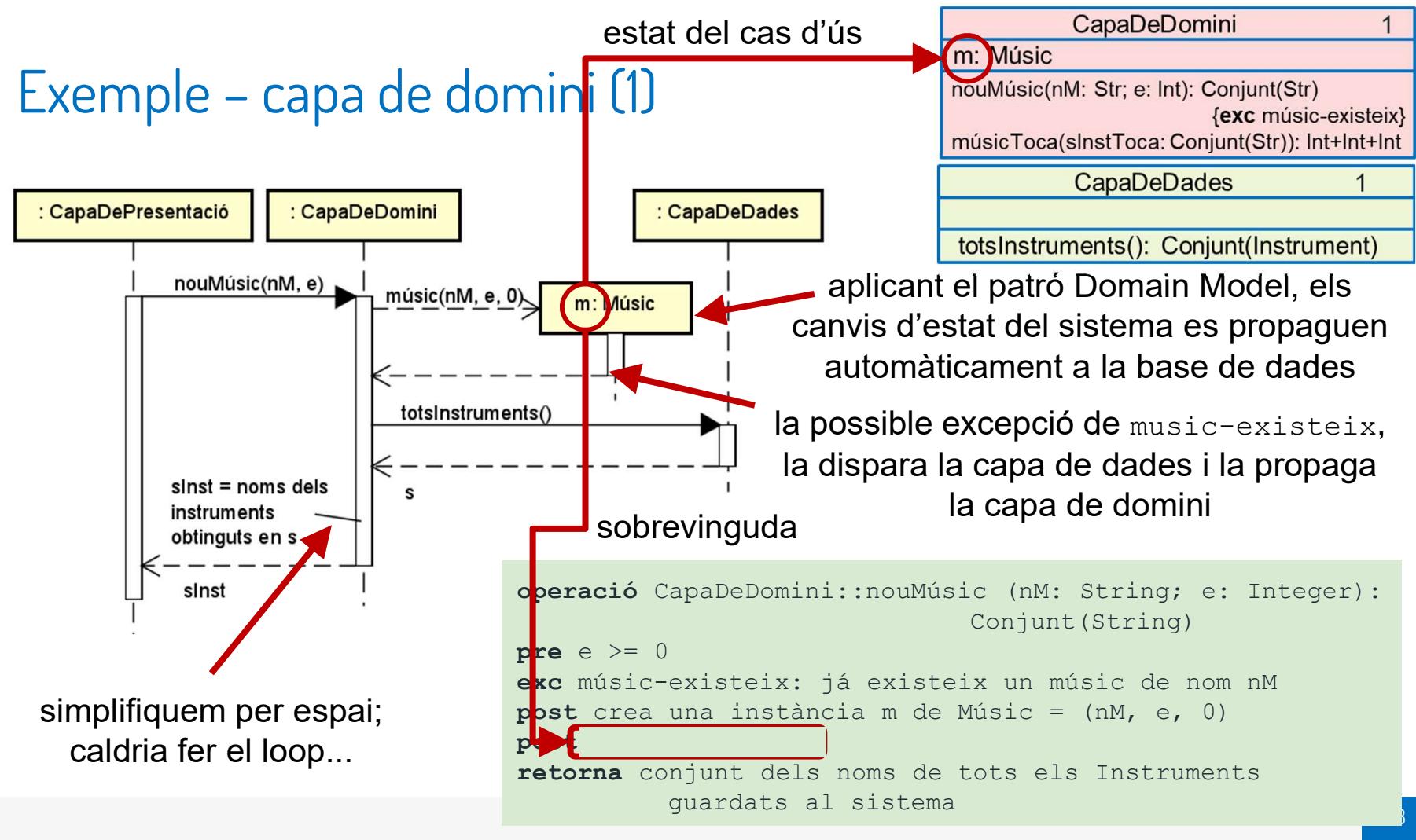
Manté la correspondència entre un model de dades O.O. i un esquema de base de dades relacional

Conceptualment, existeix un objecte únic a la capa de dades que ofereix tres operacions de consulta:

CapaDeDades	1
només si C té clau	<p>obtéC(k: T): C retorna la instància de C amb clau k {exc C-no-existeix}</p> <p>existeixC?(k: T): Bool retorna si existeix una instància de C amb clau k</p> <p>totsC(): Conjunt(C) retorna el conjunt d'instàncies de la classe C</p>

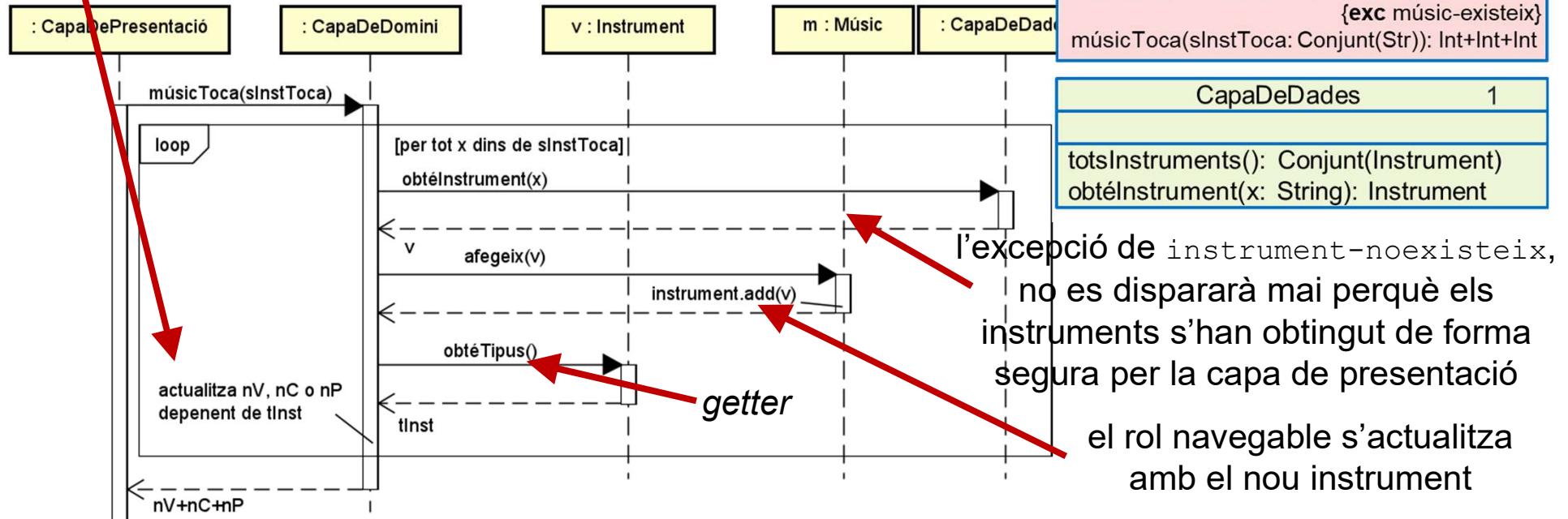
Els canvis d'estat (actualitzacions) es propaguen automàticament de la capa de domini a la capa de dades

Exemple - capa de domini (1)



atributs auxiliars; s'inclouen per motius d'eficiència

Exemple – capa de domini (2)



operació CapaDeDomini::músicToca (sInstToca: Conjunt(String)): Integer + Integer + Integer
pre per tot x dins de sInstToca: existeix un instrument amb nom x
pre self.m no toca cap instrument
post per tot x dins de sInstToca: s'enregistra que m toca x
retorna el total d'instruments de cada tipus que toca el músic: vent, corda i percussió

3.6 Simplificació del model

Casos d'ús	Diagrama de classes	Diagrames de seq.	Contractes	Disseny interfície
Concrets	Sense limitacions. <u>Operacions de capa</u>	<u>Un objecte per capa</u>	<u>De capes. Auto-continguts.</u> <u>Precs. i excs.</u>	Existeix
3. Simplificació del model				
Concrets	<u>Sense elements no suportats.</u> Operacions de capa	Un objecte per capa	De capes <u>(en la capa de domini, adaptats)</u> . Auto-continguts. Precs. i excs.	Existeix

Context

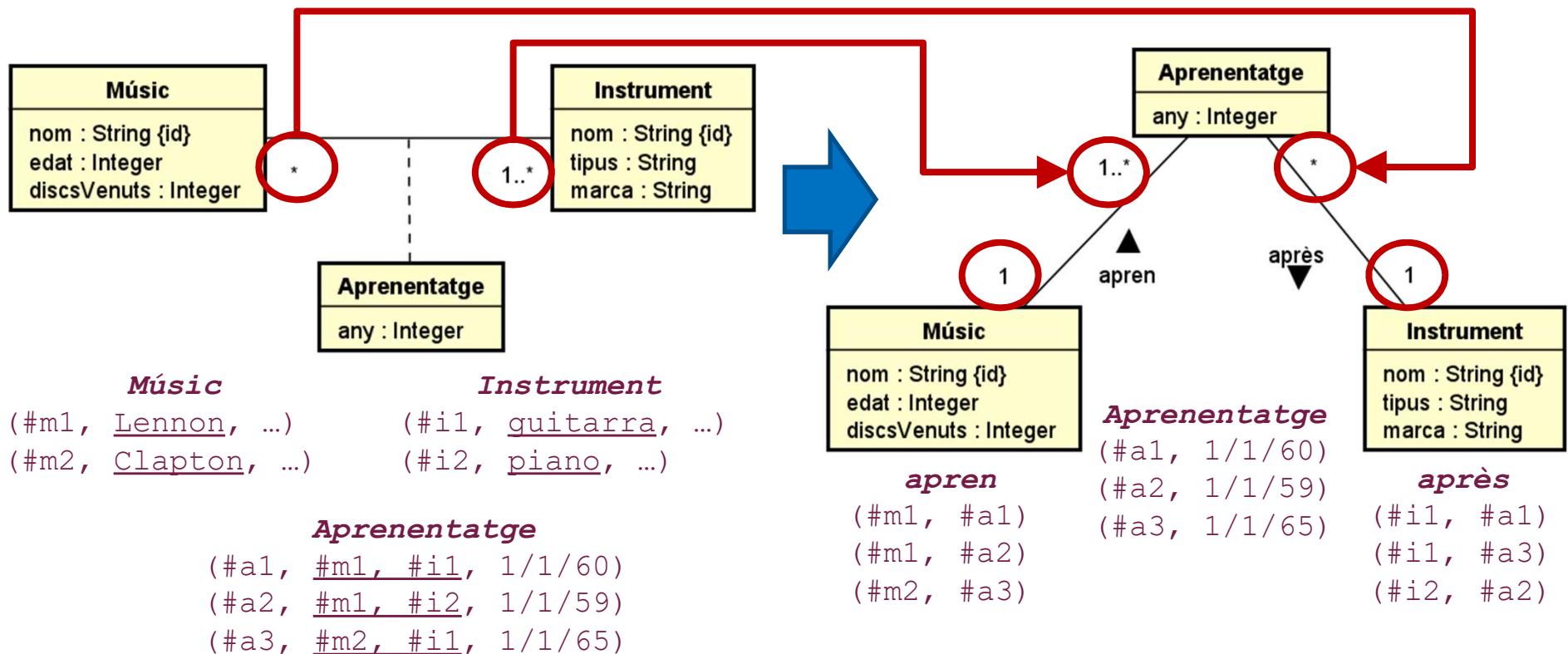
Les limitacions tecnològiques no permeten conservar tots els elements que surten al model de dades:

- classes associatives
- associacions ternàries
- especialitzacions overlapping
- atributs derivats

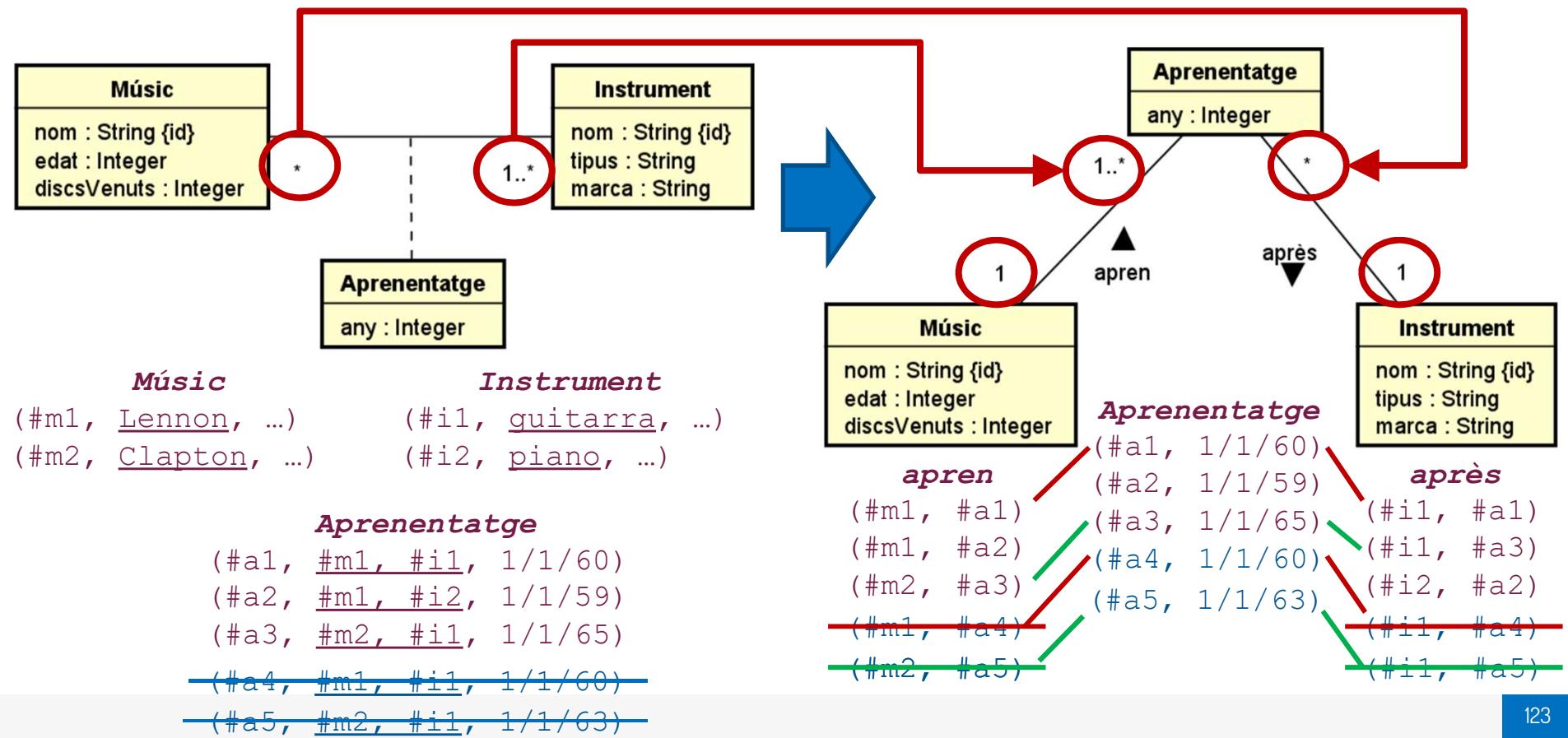
Cal transformar el model en un altre d'equivalent, substituint aquests constructors per altres, i també:

- afegint restriccions textuales noves
- modificant els contractes afectats

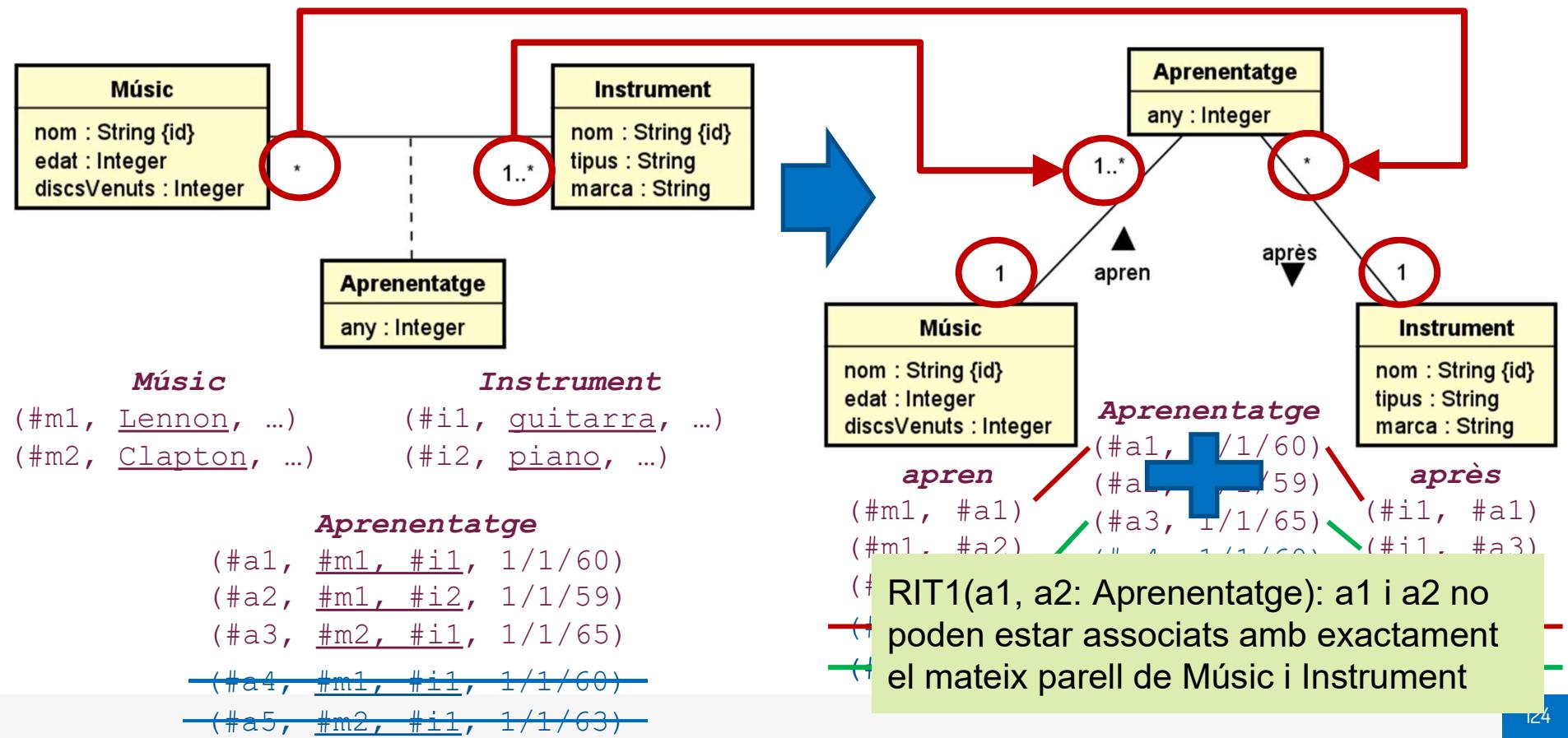
Transformacions: classes associatives



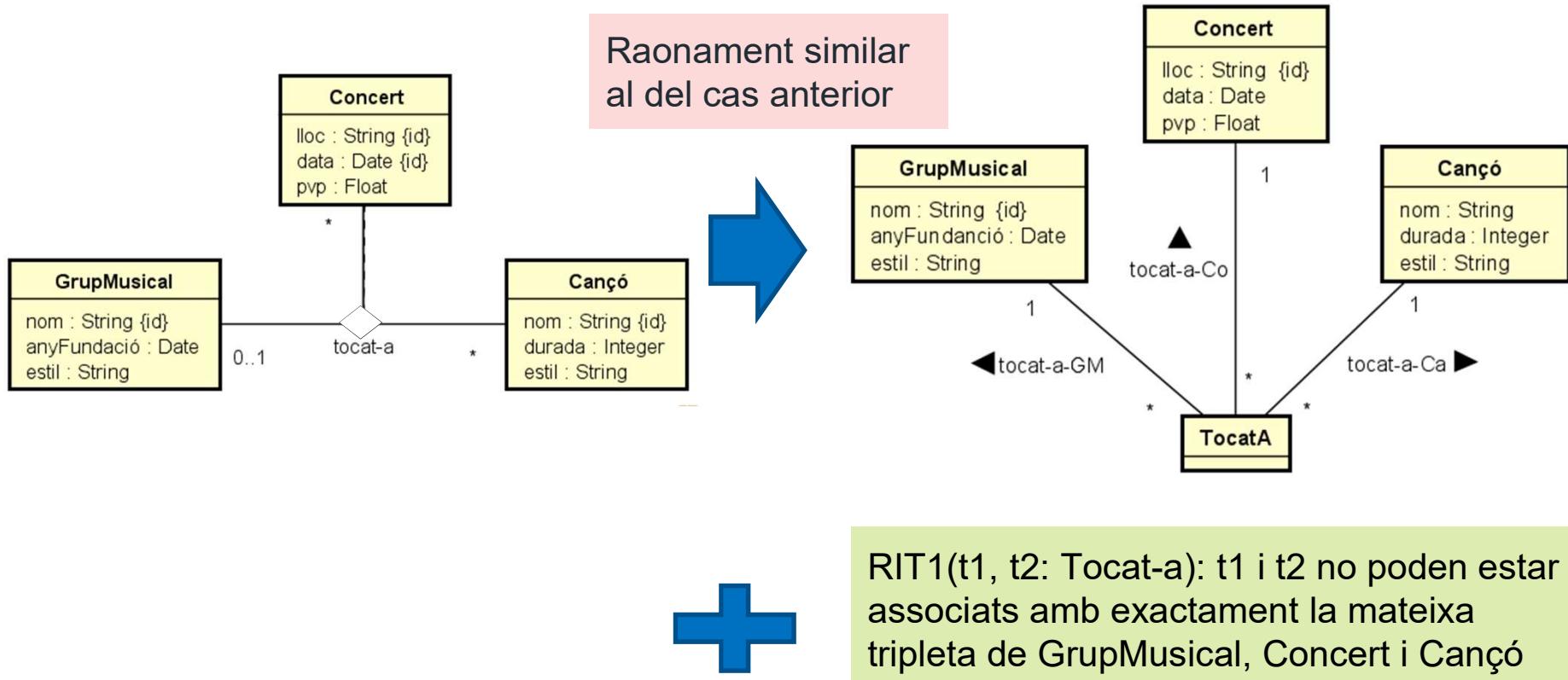
Transformacions: classes associatives



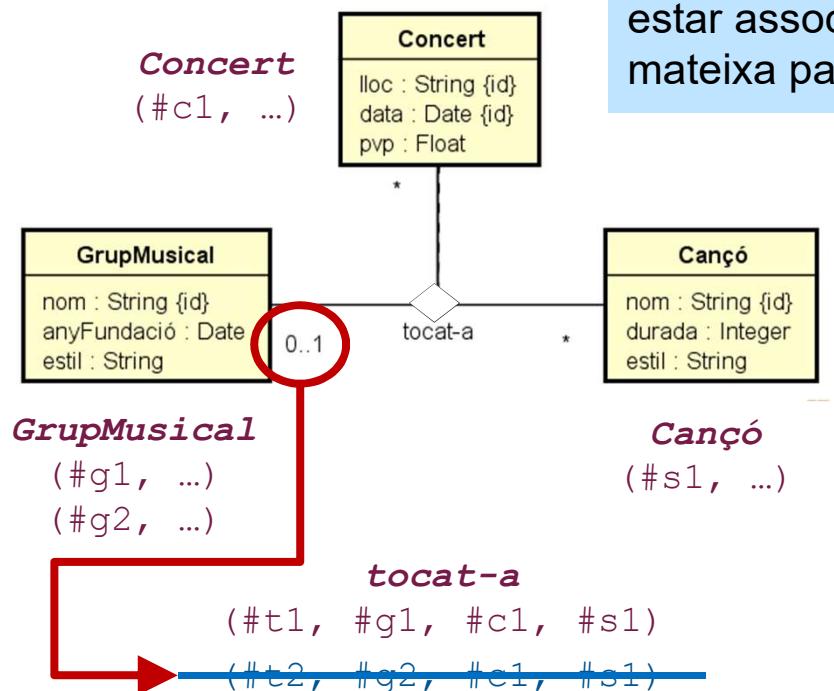
Transformacions: classes associatives



Transformacions: classes ternàries (1)

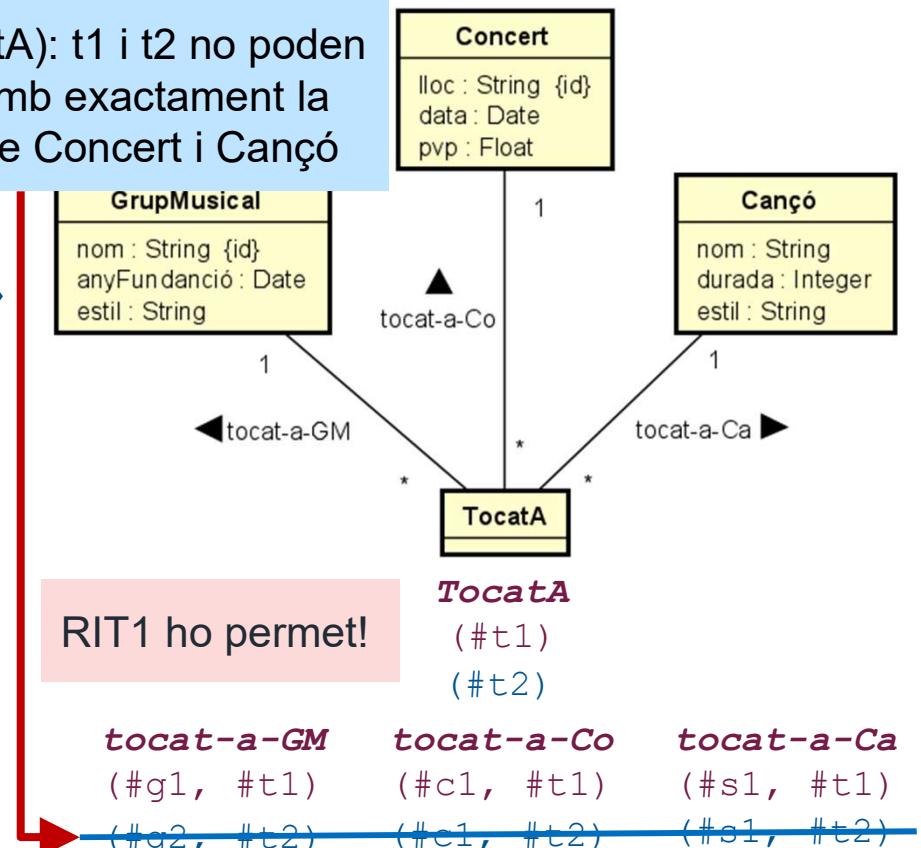


Transformacions: classes ternàries (2)

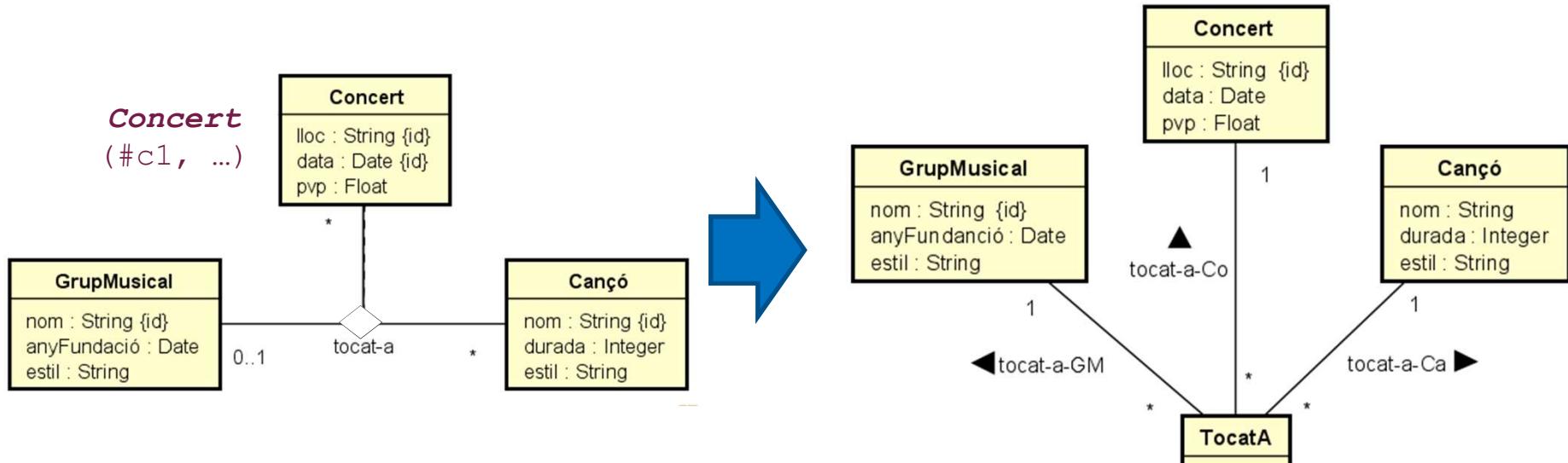


RIT2(t1, t2: TocatA): t1 i t2 no poden estar associats amb exactament la mateixa parella de Concert i Cançó

RIT1(t1, t2: TocatA): t1 i t2 no poden estar associats amb exactament la mateixa tripla de GrupMusical, Concert i Cançó



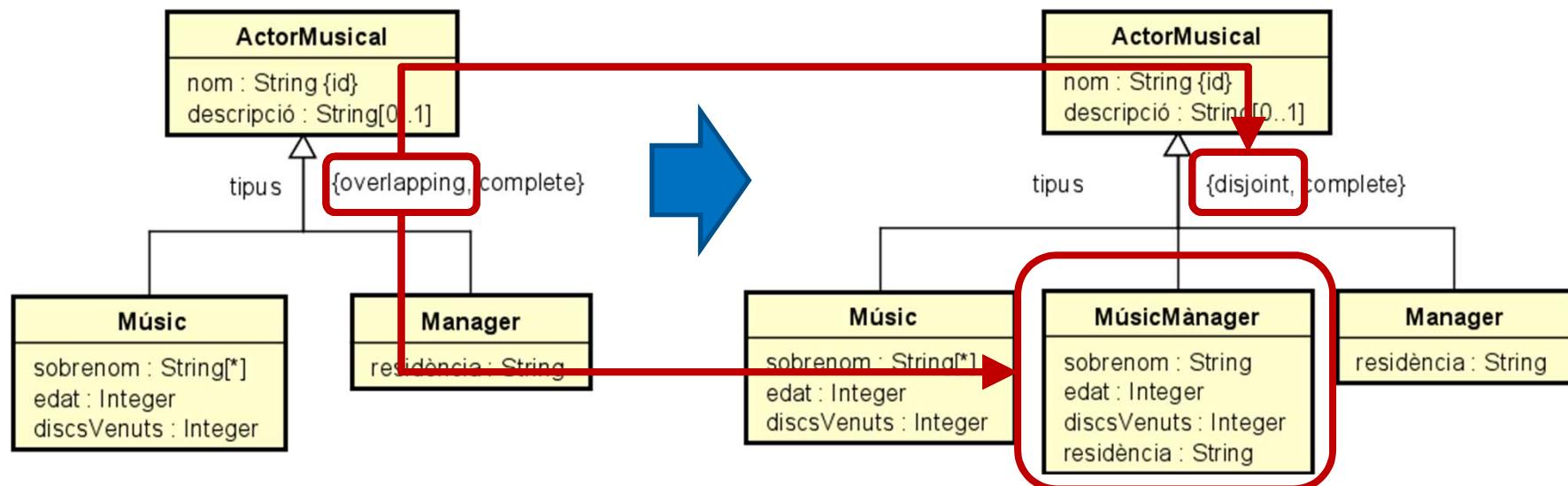
Transformacions: classes ternàries (3)



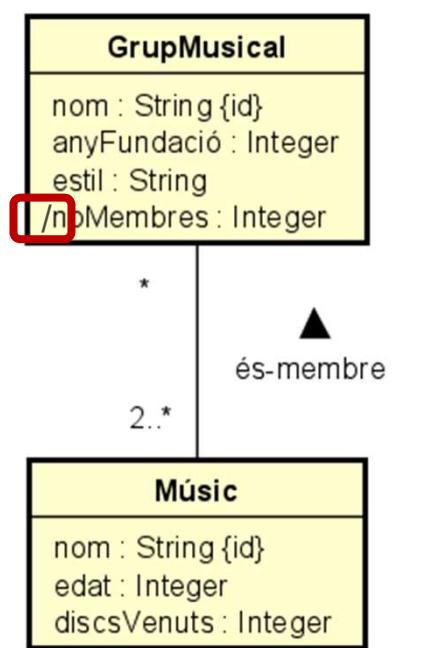
RIT2(t1, t2: Tocata): t1 i t2 no poden estar associats amb exactament la mateixa parella de Concert i Cançó

~~RIT1(t1, t2: Tocata): t1 i t2 no poden estar associats amb exactament la mateixa tripla de GrupMusical, Concert i Cançó~~

Transformacions: especialitzacions *overlapping*



Transformacions: atributs derivats (model de dades)



opció 1



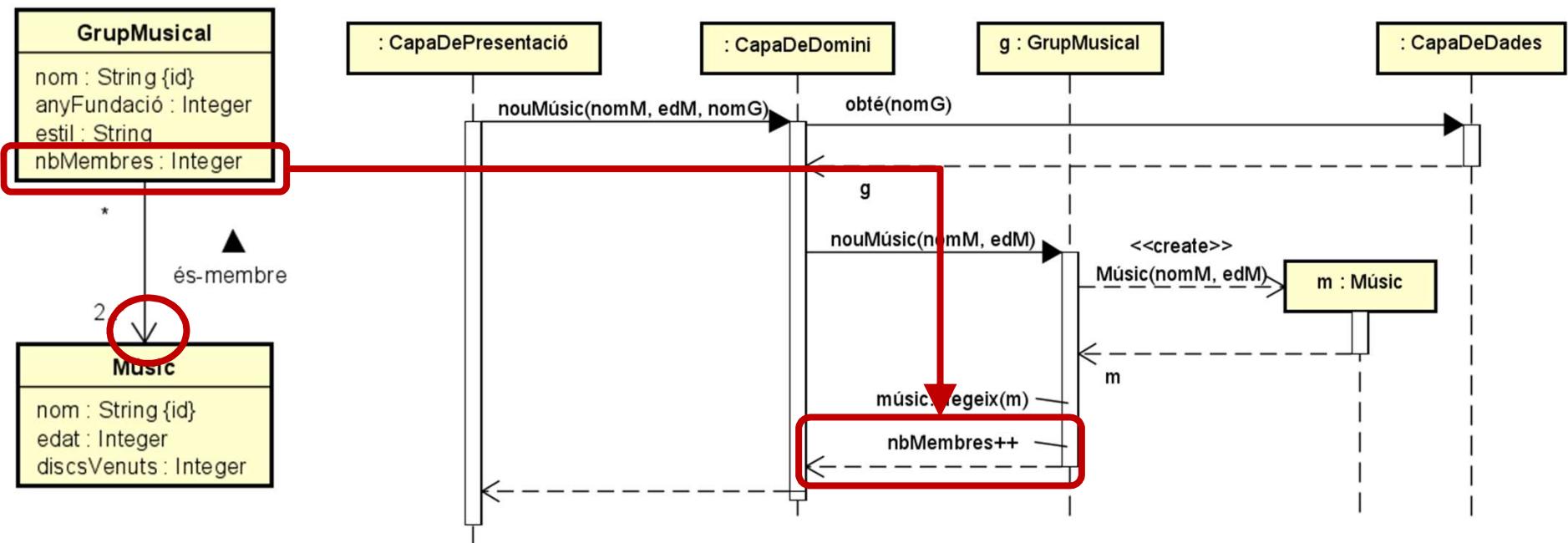
atribut materialitzat

opció 2



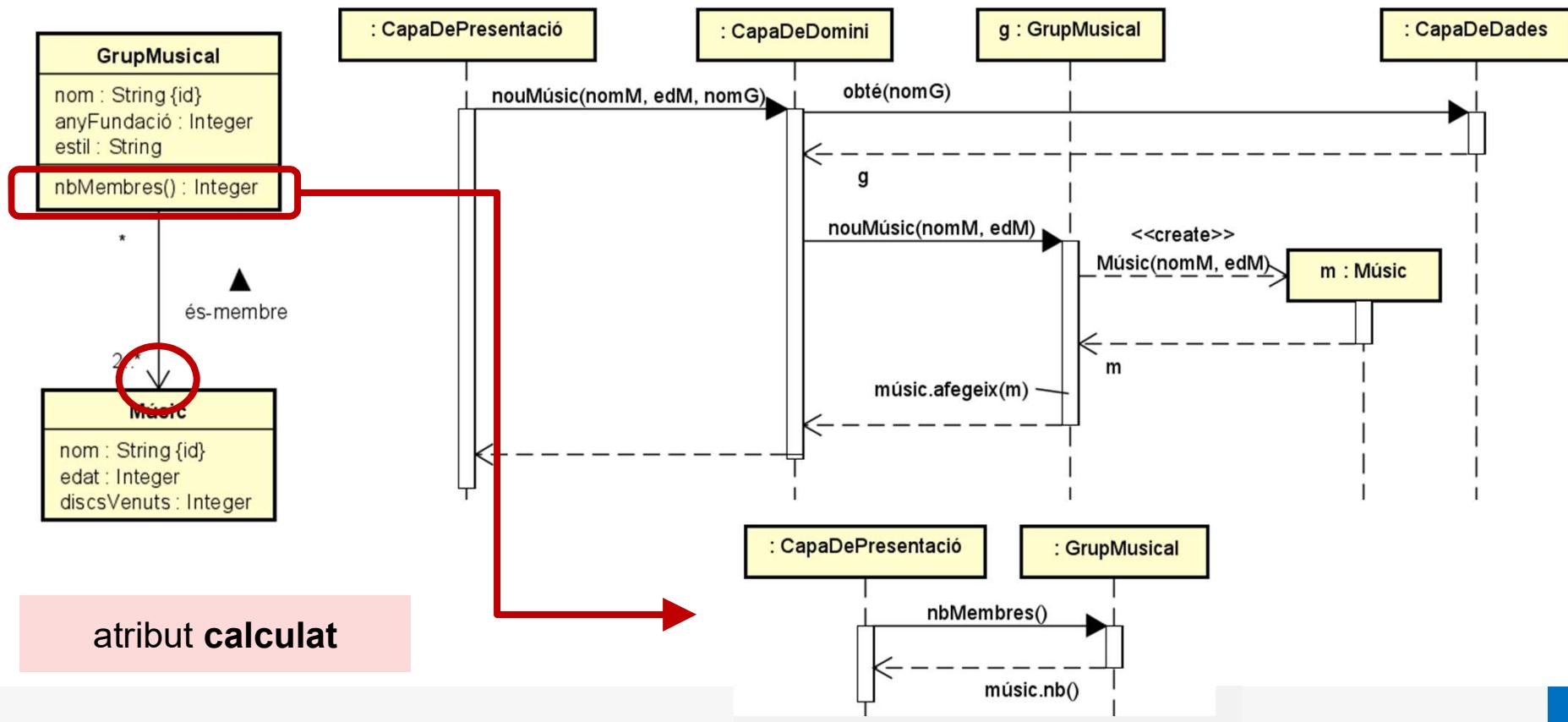
atribut calculat

Transformacions: atributs derivats (model del comportament)



atribut materialitzat

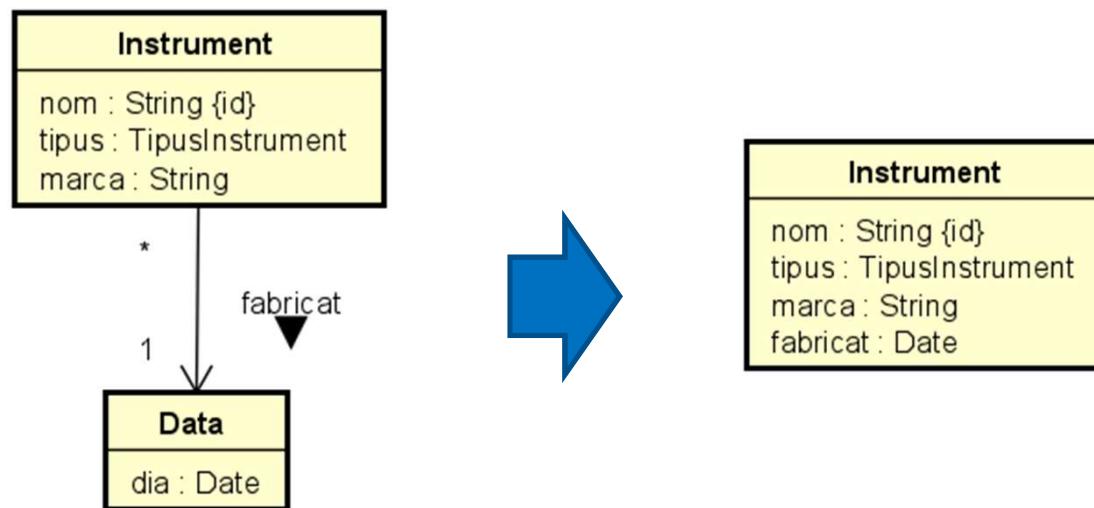
Transformacions: atributs derivats (model del comportament)



Transformacions: classe Data

La classe Data (o similars) venen formalment com a part del llenguatge de programació:

- convertim les classes en atributs, respectant la multiplicitat



3.7 Disseny de les operacions

	Casos d'ús	Diagrama de classes	Diagrames de seq.	Contractes	Disseny interfície
	Concrets	<u>Sense elements no suportats.</u> Operacions de capa	Un objecte per capa	De capes (<u>en la capa de domini, adaptats</u>). Auto-continguts. Precs. i excs.	Existeix
4. Disseny de les operacions					
	Concrets	Sense elements no suportats. <u>Operacions de classe</u>	<u>Tants objectes com calgui</u>	<u>De classe, incloent-ne operacions auxiliars.</u> Auto-continguts. Precs. i excs.	Existeix

Context

Partim d'un model que:

- té un model de dades simplificat, preparat per codi O.O.
- té un model del comportament adaptat a aquests canvis, tal que:
 - cada capa té definides les seves operacions amb contractes auto-continguts
 - el patró dominant a la capa de domini, i el patró corresponent a la capa de dades, estan ja determinats

Per finalitzar el disseny, cal finalitzar les tres capes:

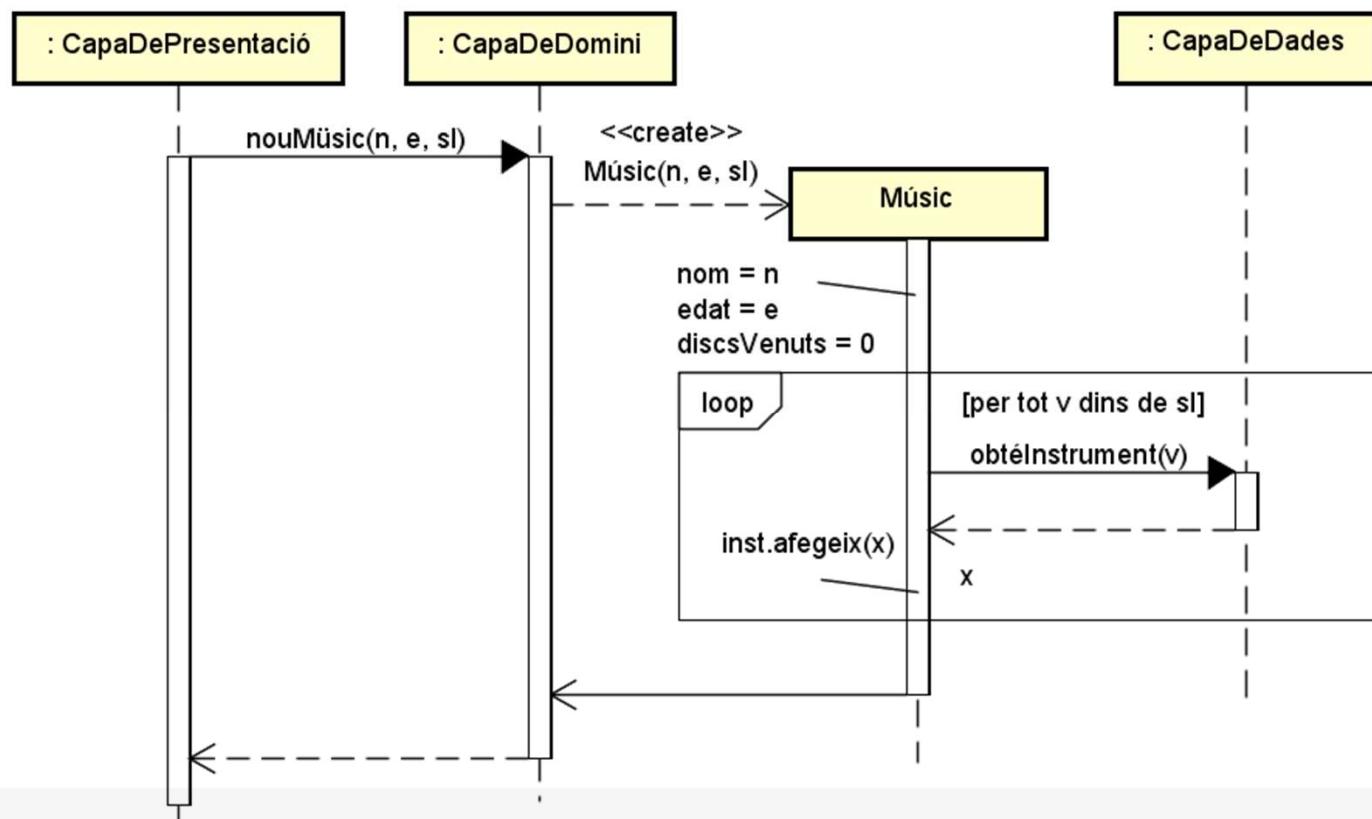
- capa de presentació: ja està fet (v. Assignació de Responsabilitats)
- capa de dades: en Domain Model + Data Mapper, les operacions estan predefinides
- capa de domini: és el nostre objectiu!

Disseny de la capa de domini

Per cada esdeveniment extern:

- hi ha un objecte (instància d'una classe anomenada Controlador) que el rep
- aquest objecte busca quin(s) objecte(s) poden col.laborar per satisfer el contracte
 - assignació de responsabilitats a objectes (ARO)
 - el diagrama de seqüència reflecteix aquesta col.laboració
 - quan és necessari, la capa de domini interacciona amb la capa de dades per obtenir / modificar objectes

Ja ho coneixem...



Patrons de disseny

El disseny de les capes ha de seguir aplicant bones pràctiques tant com sigui possible:

- disposem al nostre abast d'una col·lecció de patrons de disseny O.O.
- especialment útils en dissenyar la capa de domini
 - degut a la seva complexitat més gran
- es caracteritzen per la tripla context-problema-solució
- en aquesta assignatura, ens centrem en uns pocs patrons representatius

Patró Controlador (*Controller*)

Context:

- un component software (subsistema, capa, ...) rep esdeveniments
- un cop interceptat, aquest esdeveniment ha de ser tractat

Problema:

- quin objecte té la responsabilitat de rebre i tractar l'esdeveniment?

Solució:

- un objecte específic, anomenat **controlador**, rep l'esdeveniment
- el controlador queda sota el control d'un altre objecte
- el controlador delega en altres objectes el tractament de l'esdeveniment
- aquests altres objectes desconeixen l'existència del controlador

Tipus de controladors

En l'arquitectura en capes, els controladors s'usen sobretot per rebre i tractar els esdeveniments que arriben a les capes

Trobem tres grans variants de controladors:

- **façana**: un únic controlador per capa
- **cas d'ús**: un controlador per cada cas d'ús
- **transacció**: un controlador per cada (instància d') esdeveniment

Controlador Façana (*Façade*)

Ja el coneixem...

Un únic objecte rep tots els esdeveniments de la capa

- instància d'una classe *singleton*
- una operació per esdeveniment
- atributs necessaris per guardar estat intermedi dels casos d'ús

CapaDePresentació	1
altaMúsicInici()	
altaMúsicPrimerOK()	

CapaDeDomini	1
m: Músic	
nV, nC, nP: Integer	
nouMúsic(nM: Str; e: Int): Conjunt(Str)	
{exc músic-existeix}	
músicToca(sInstToca: Conjunt(Str)): Int+Int+Int	

CapaDeDades	1
totsInstruments(): Conjunt(Instrument)	
obtéInstrument(x: String): Instrument	

Controlador Cas d'Ús

Aplicable a les capes de presentació i/o de domini

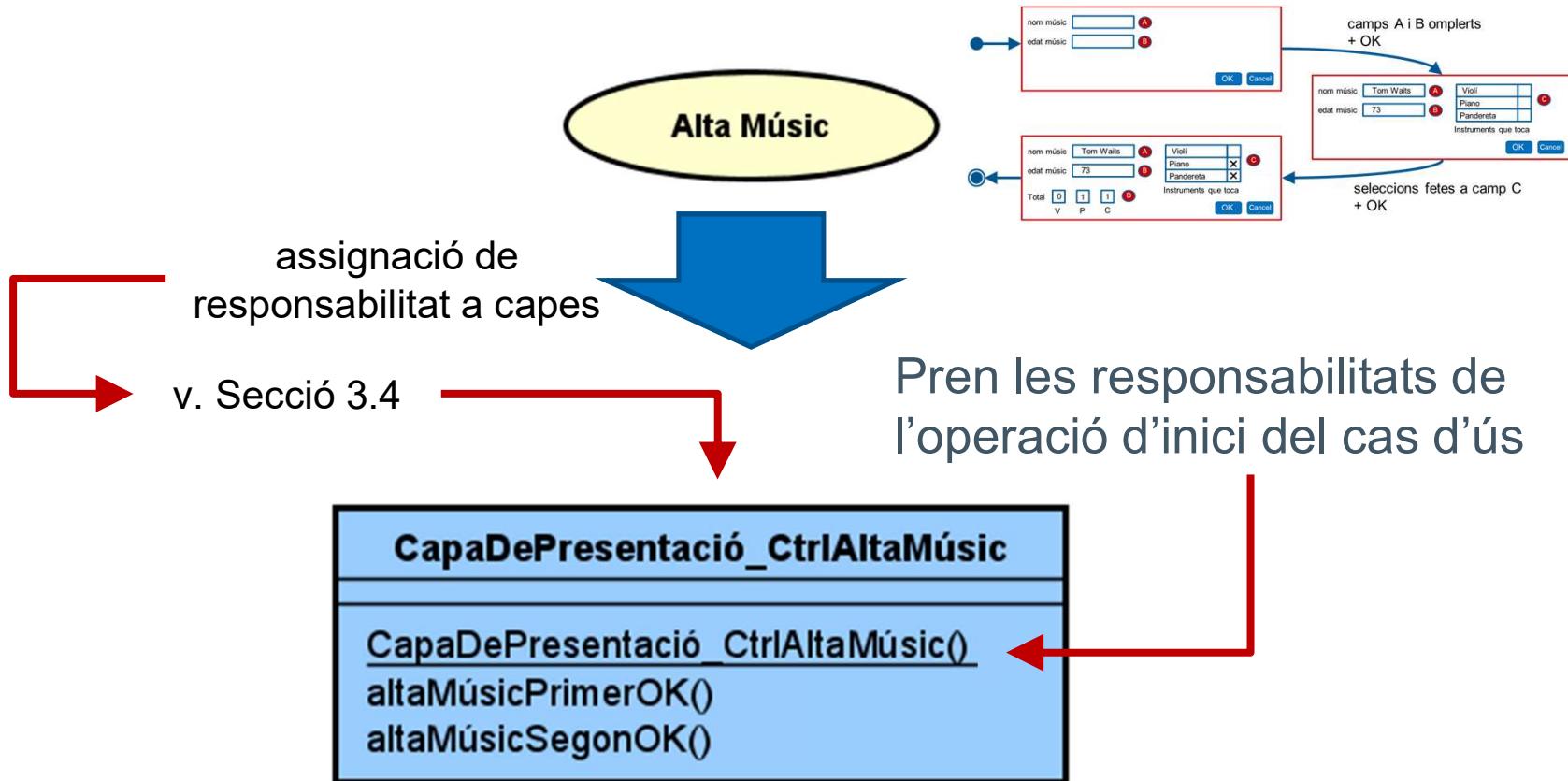
S'associa un controlador a cada cas d'ús:

- cada cas d'ús *CU* dóna lloc a una classe *CtrlCU*
- *CtrlCU* ofereix una operació per cada esdeveniment de *CU* (de presentació o extern, dependent de la capa)
- *CtrlCU* inclou els atributs necessaris per preservar informació durant l'execució de *CU*

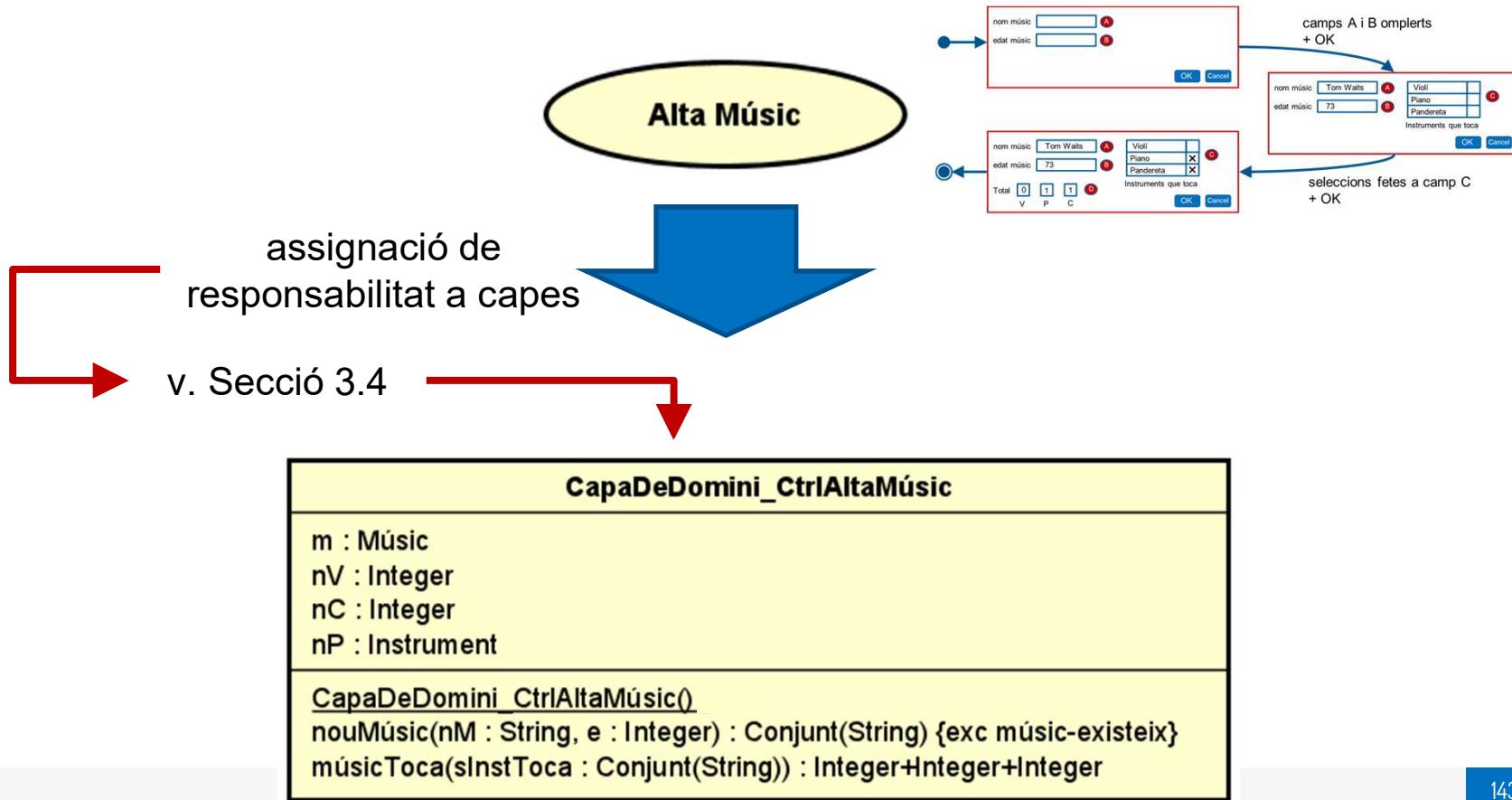
En començar l'execució d'un cas d'ús *CU*:

- es crea un objecte *x* instància de la classe *CtrlCU*
 - aquesta operació enregistra l'inici del cas d'ús
- es van executant les operacions de *CU* segons l'ordre determinat pel model del comportament, invocant les operacions corresponents de *CtrlCU*
- en acabar l'execució de *CtrlCU*, es destrueix l'objecte *x*

Controlador Cas d'Ús – exemple, capa de presentació: model dades

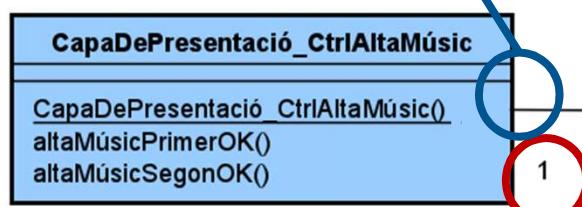


Controlador Cas d'Ús – exemple, capa de domini: model dades

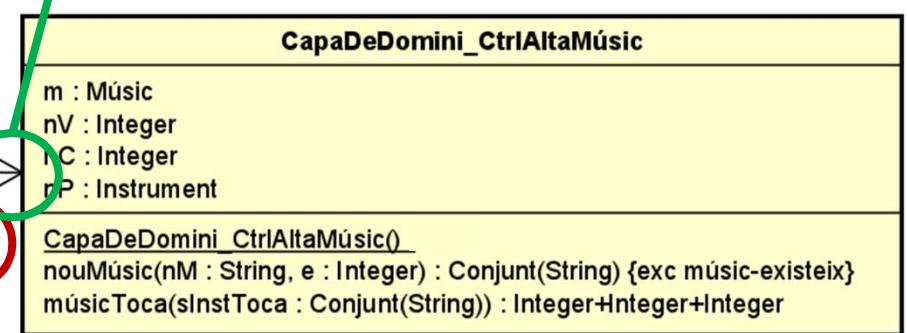


Controlador Cas d'Ús – exemple, lligant les dues capes, model dades

El controlador de domini no sap sobre el controlador de presentació



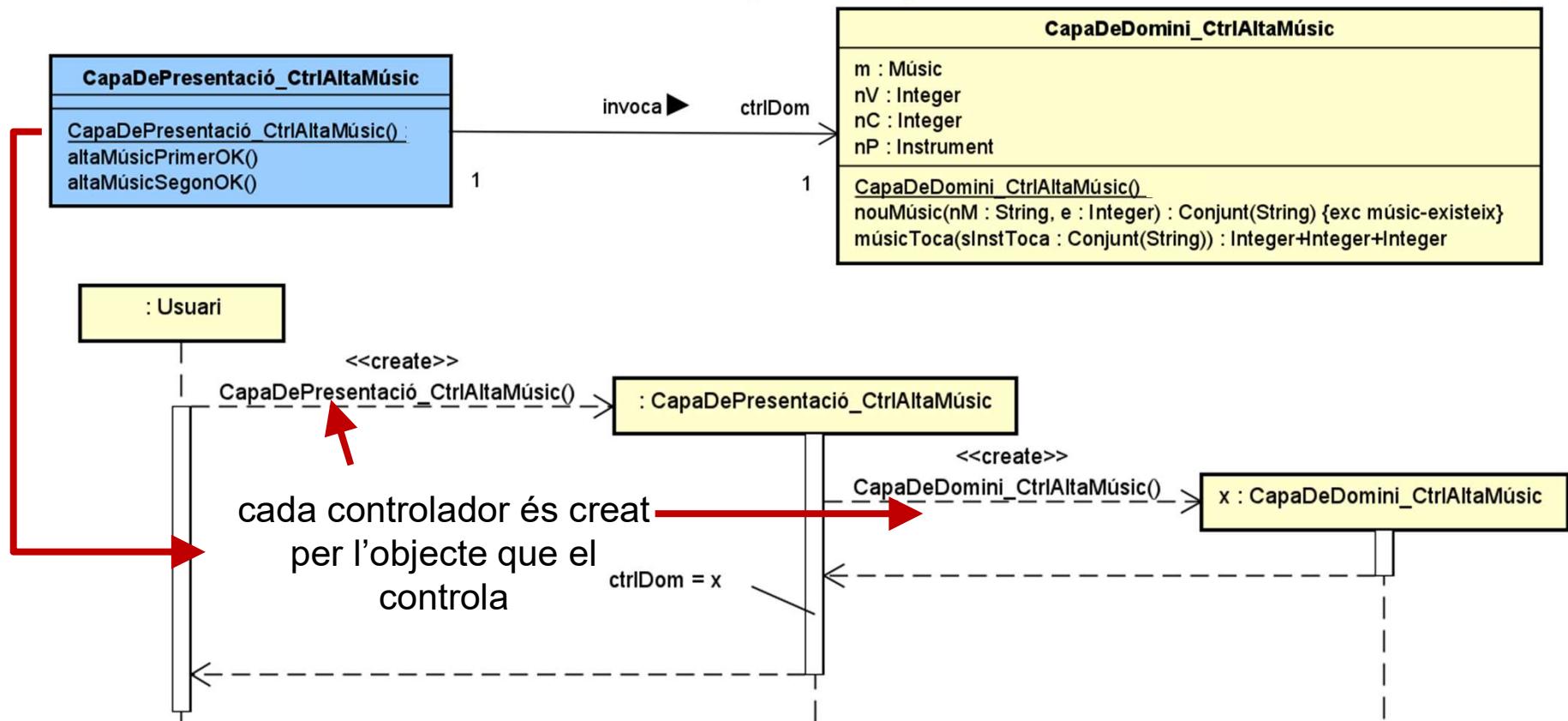
El controlador de presentació té el control sobre el controlador de domini



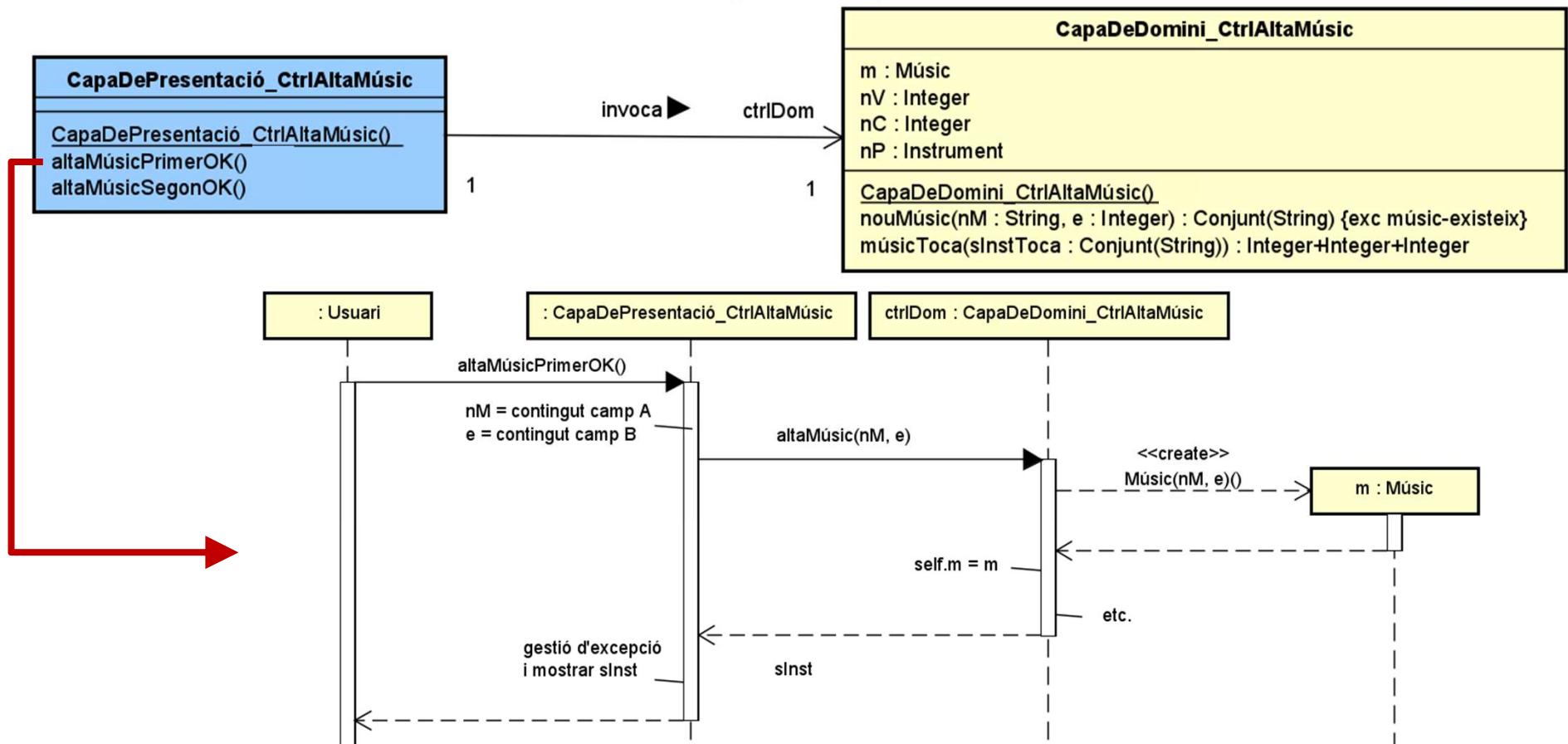
La vida dels dos controladors està totalment interlligada

- L'un no pot existir sense l'altre

Controlador Cas d'Ús – exemple, capa de domini: model comp., 1



Controlador Cas d'Ús – exemple, capa de domini: model comp., 2



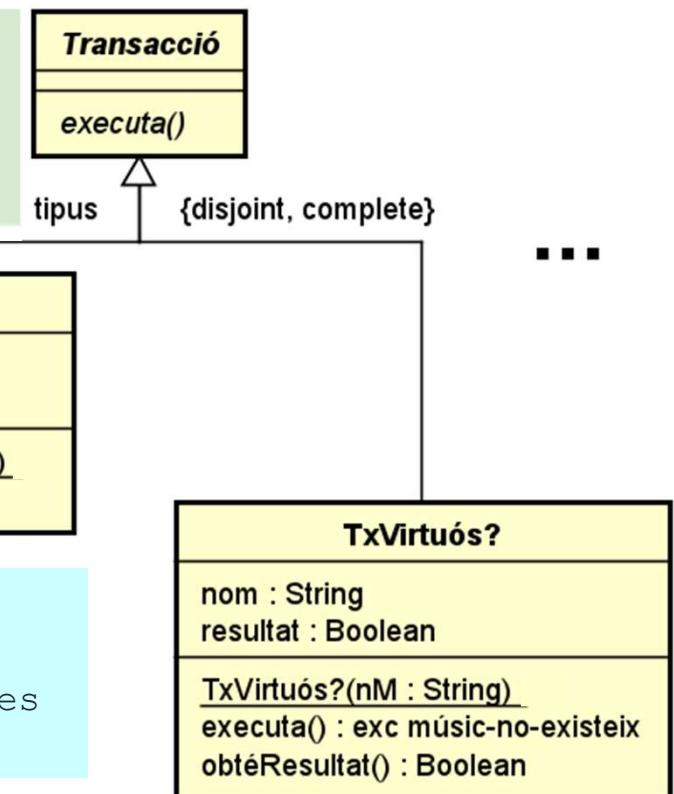
Controlador Transacció (*Transaction*)

Es crea un objecte només per tractar un esdeveniment:

- instància d'una classe associada a l'esdeveniment
- s'hi afegeix un atribut per cada paràmetre de l'operació associada
 - l'operació constructora té un paràmetre per cada atribut **in** i **in/out**
 - per als atributs **out** i **in/out**, s'afegeix una operació *getter*
- si l'operació retorna valor, s'hi afegeix un altre atribut
 - i el *getter* corresponent
- finalment, hi ha una operació **executa()** que s'encarrega d'executar l'operació en resposta a l'esdeveniment
 - heretada d'una operació abstracte compartida per tots els controladors transacció
 - declara les excepcions de l'operació associada a l'esdeveniment

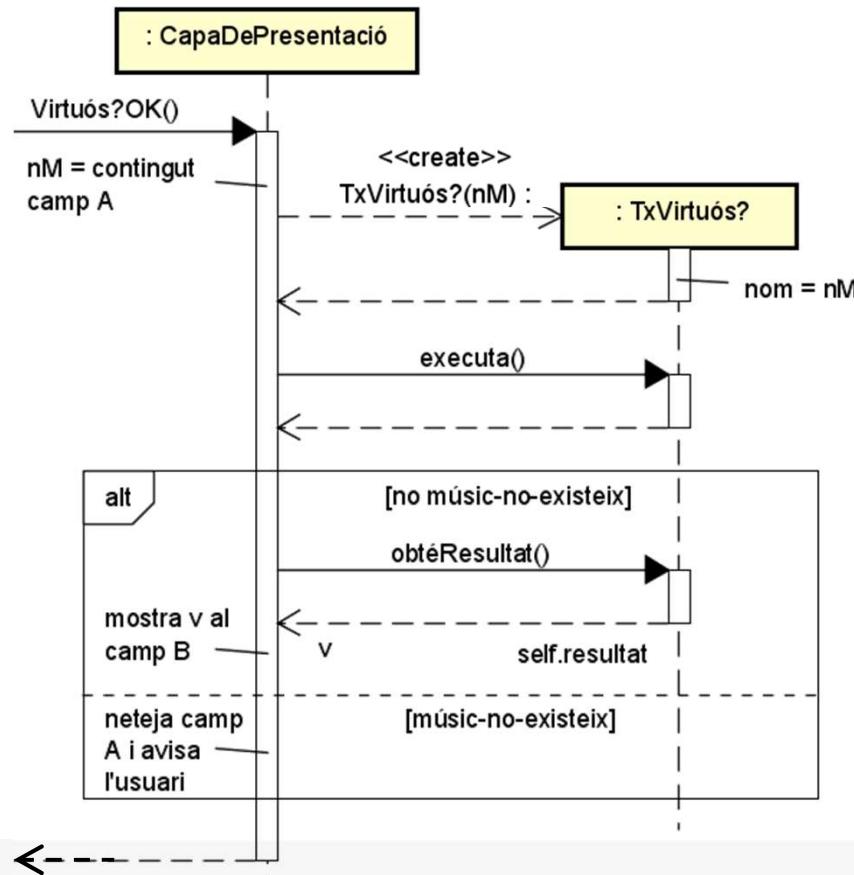
Controlador Transacció – exemple, capa de domini: model dades

```
operació nouMúsic (nM: String; e: Integer)
pre e >= 0
exc músic-existeix: já existeix um músic de nom nM
post crea una instància m de Músic = (nM, e, 0)
```



```
operació virtuós? (nM: String): Boolean
exc músic-no-existeix un músic de nom nM
retorna cert si el músic de nom nM toca els tres
tipus d'instruments
```

Controlador Transacció – exemple, capa de domini: model comp.



Comparativa dels tres tipus de controladors

	Granularitat	Acoblament	Cohesió	Comprensió	Ús
Façana	Un (per capa)	Alt	Baixa	Molt alta	Trivial
Cas d'ús	Un per cas d'ús (per capa)	Mig	Alta	Alta	No trivial
Transacció	Un per cada operació de cada cas d'ús	Baix	Alta	Mitjana	Complicat

Patró Plantilla (*Template*)

Context:

- Una operació op definida en una superclasse C d'una jerarquia té un comportament diferent en les seves diferents subclasses C_1, \dots, C_n
- Totes les diferents versions d' op tenen part del seu comportament comú

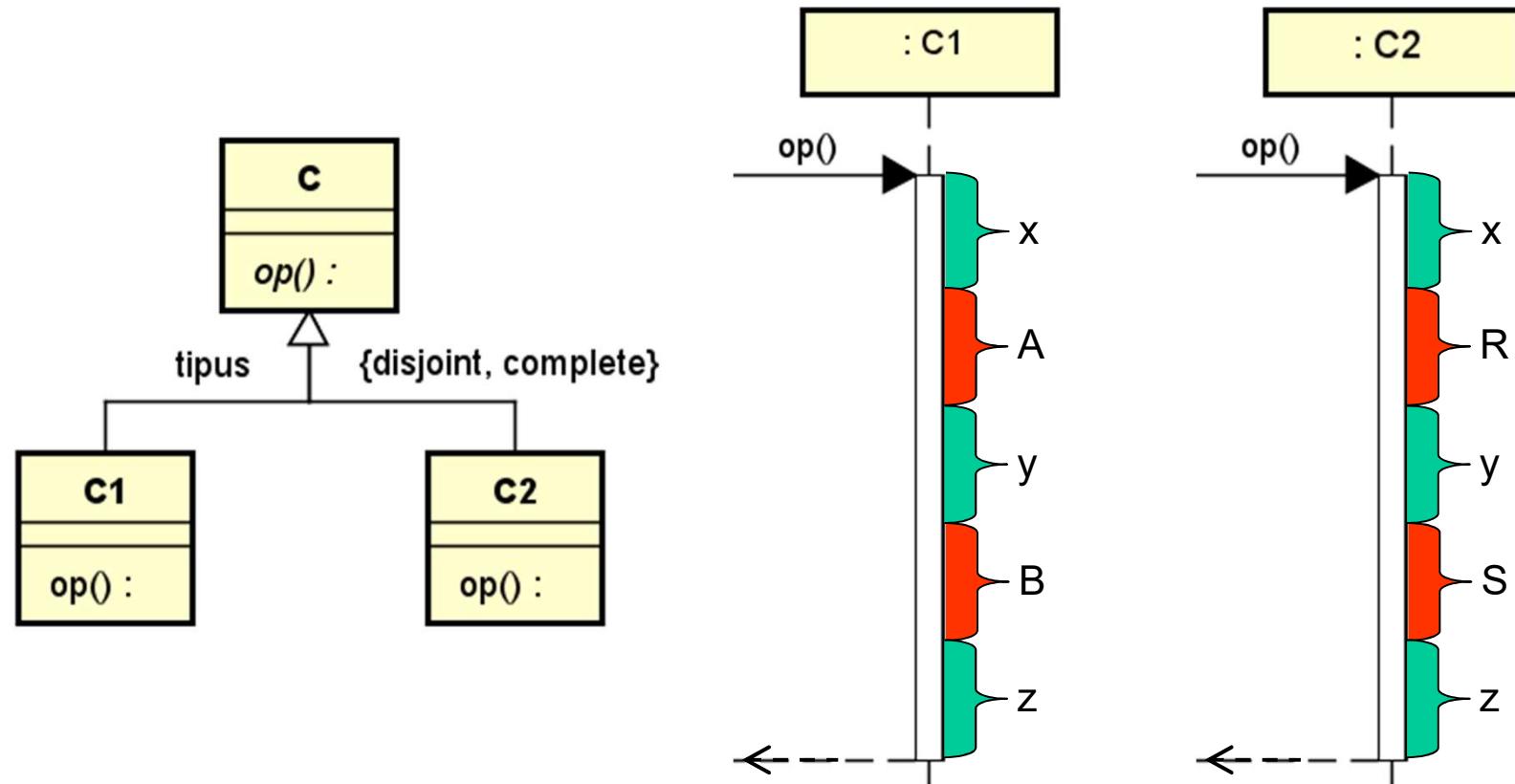
Problema:

- com evitar repetir el comportament comú a cada classe C_1, \dots, C_n ?

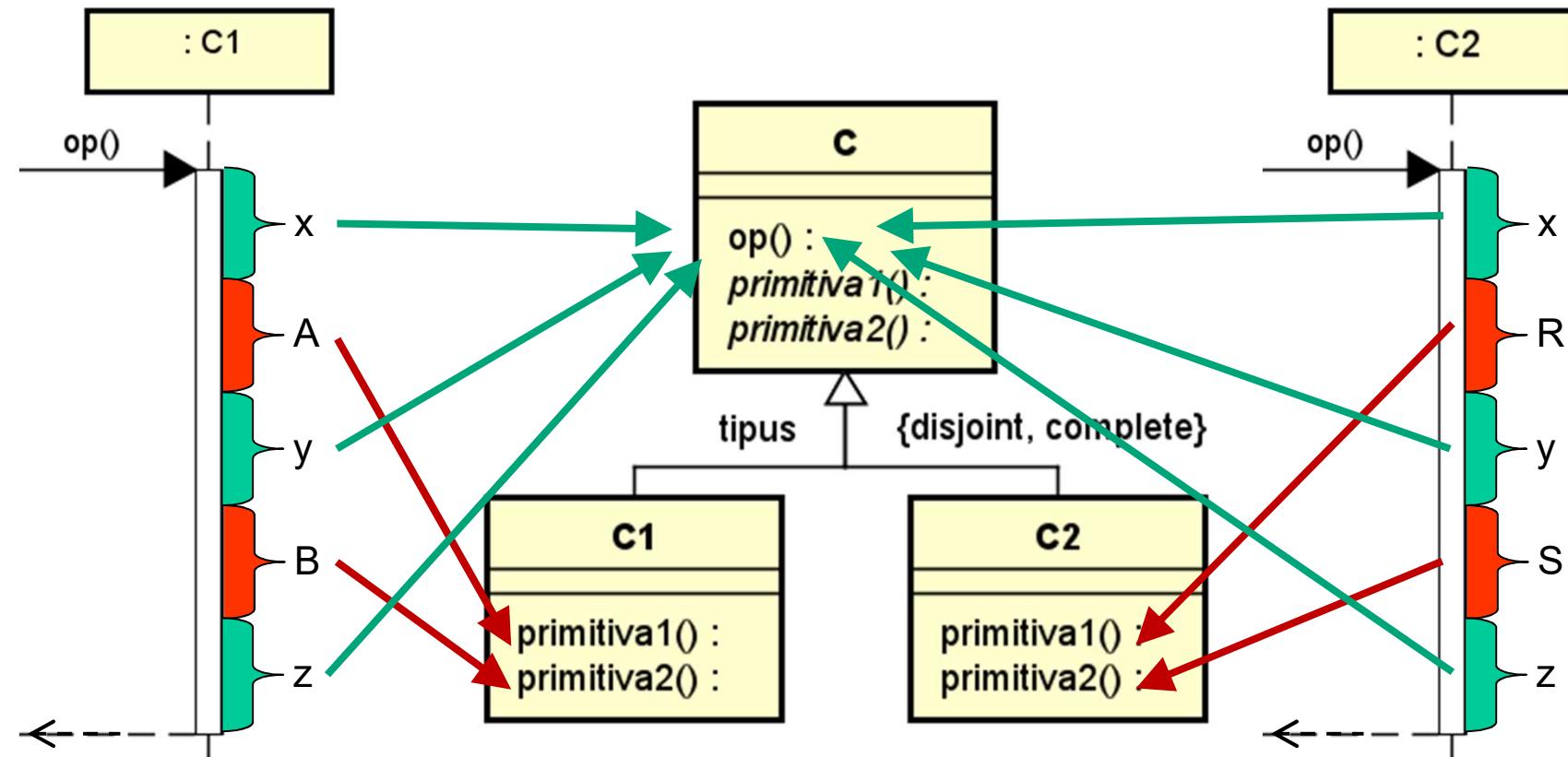
Solució:

- definir op com a abstracte a C implementant el comportament comú
- a cada C_k , implementar el comportament específic a l'operació concreta op
- connectar ambdues operacions usant polimorfisme i vinculació dinàmica

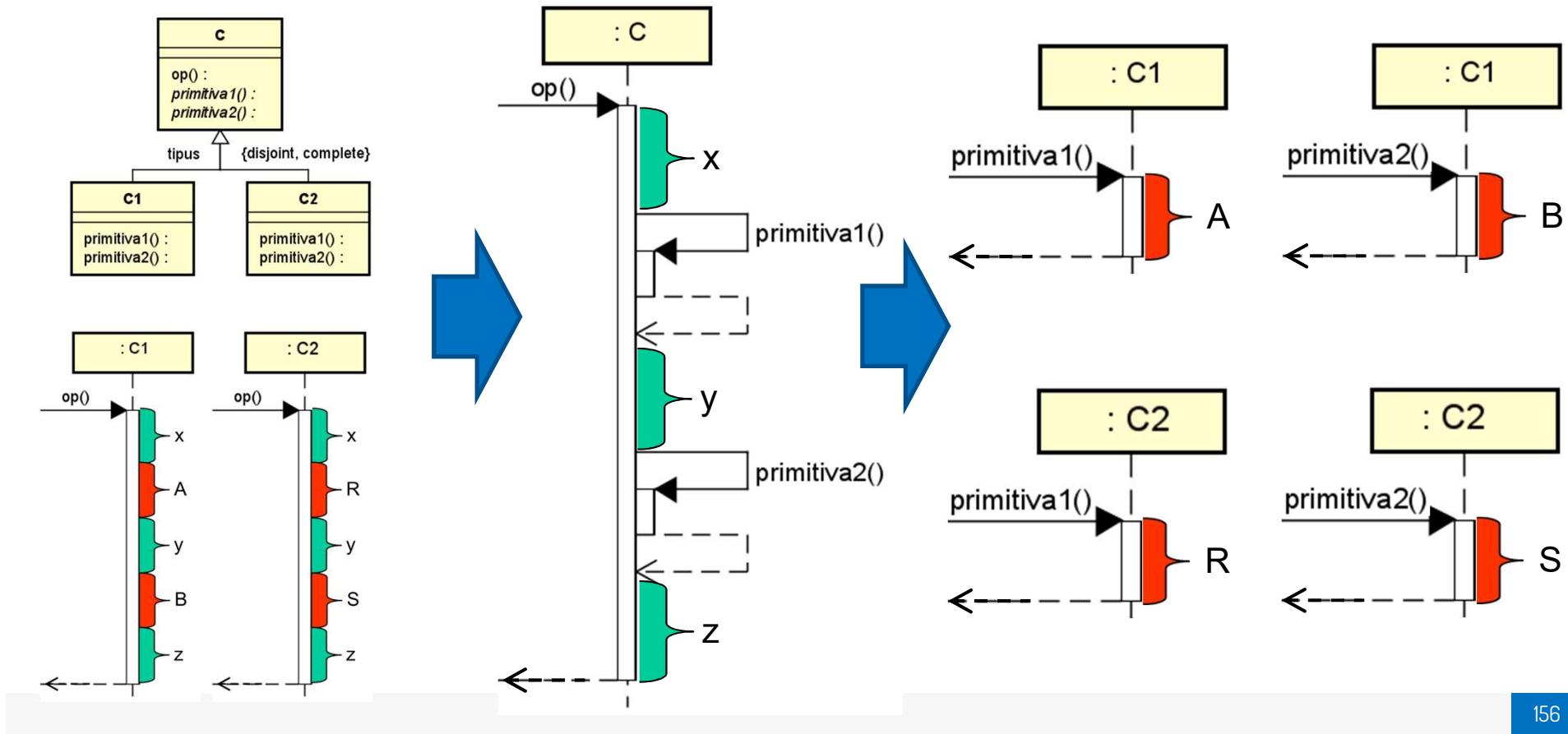
Patró Plantilla – context en detall



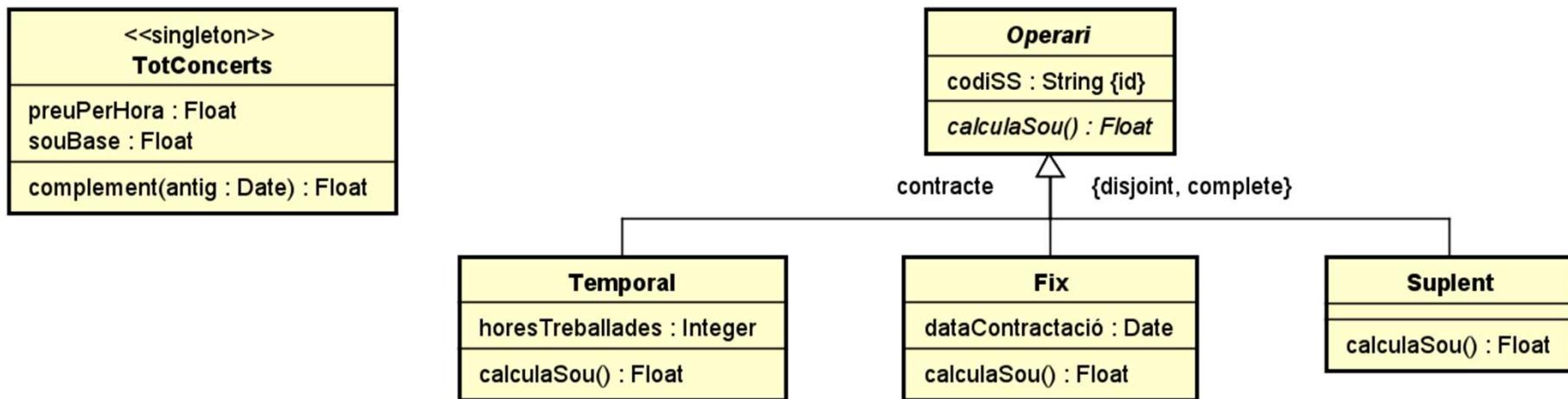
Patró Plantilla – solució, model de dades



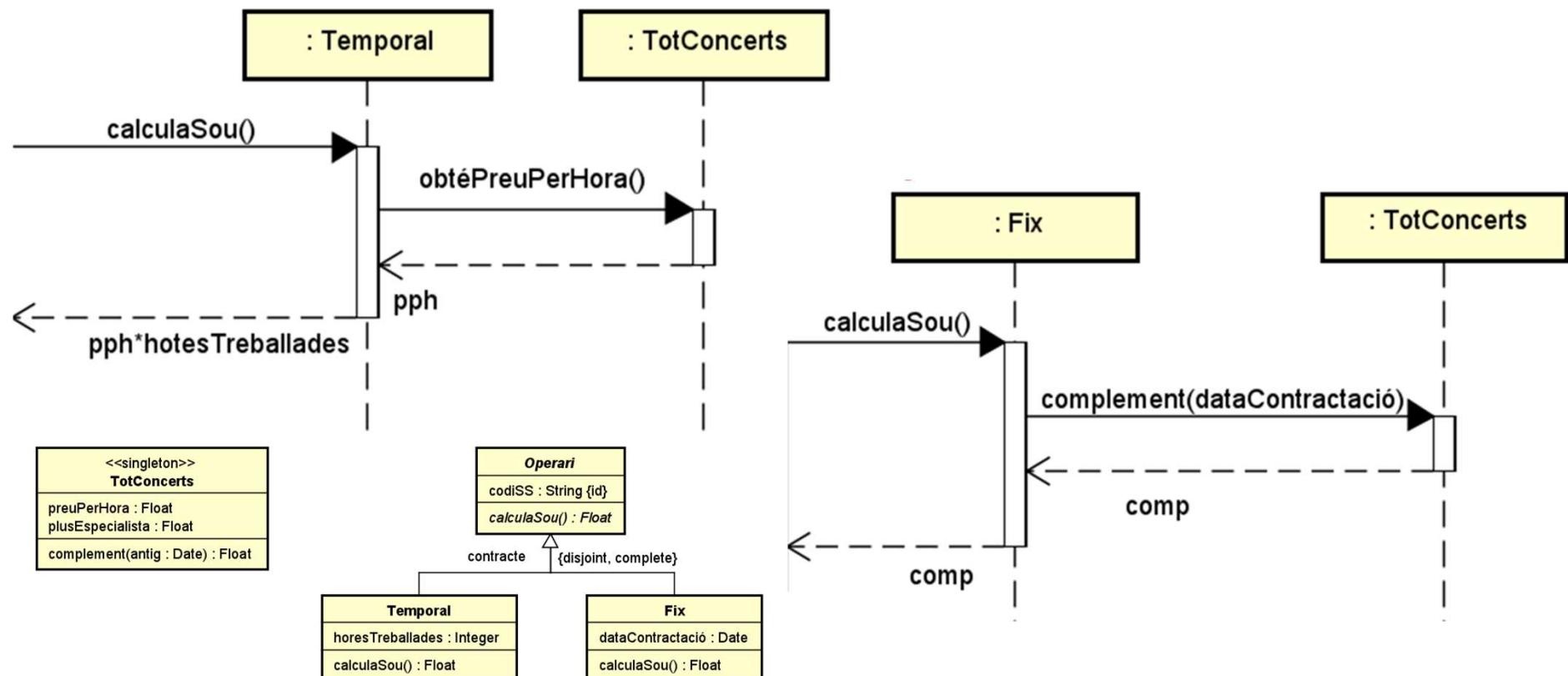
Patró Plantilla – solució, model del comportament



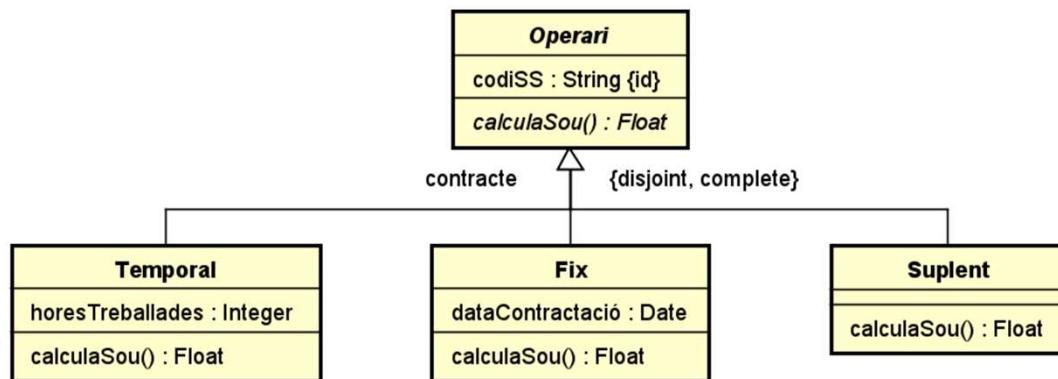
Patró Plantilla – exemple (revisitat)



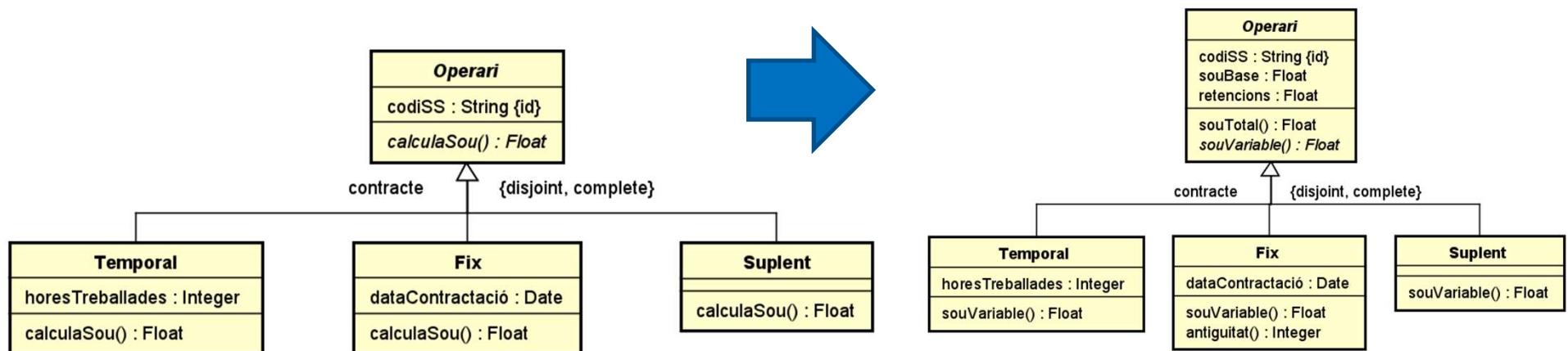
Patró Plantilla – exemple (revisitat)



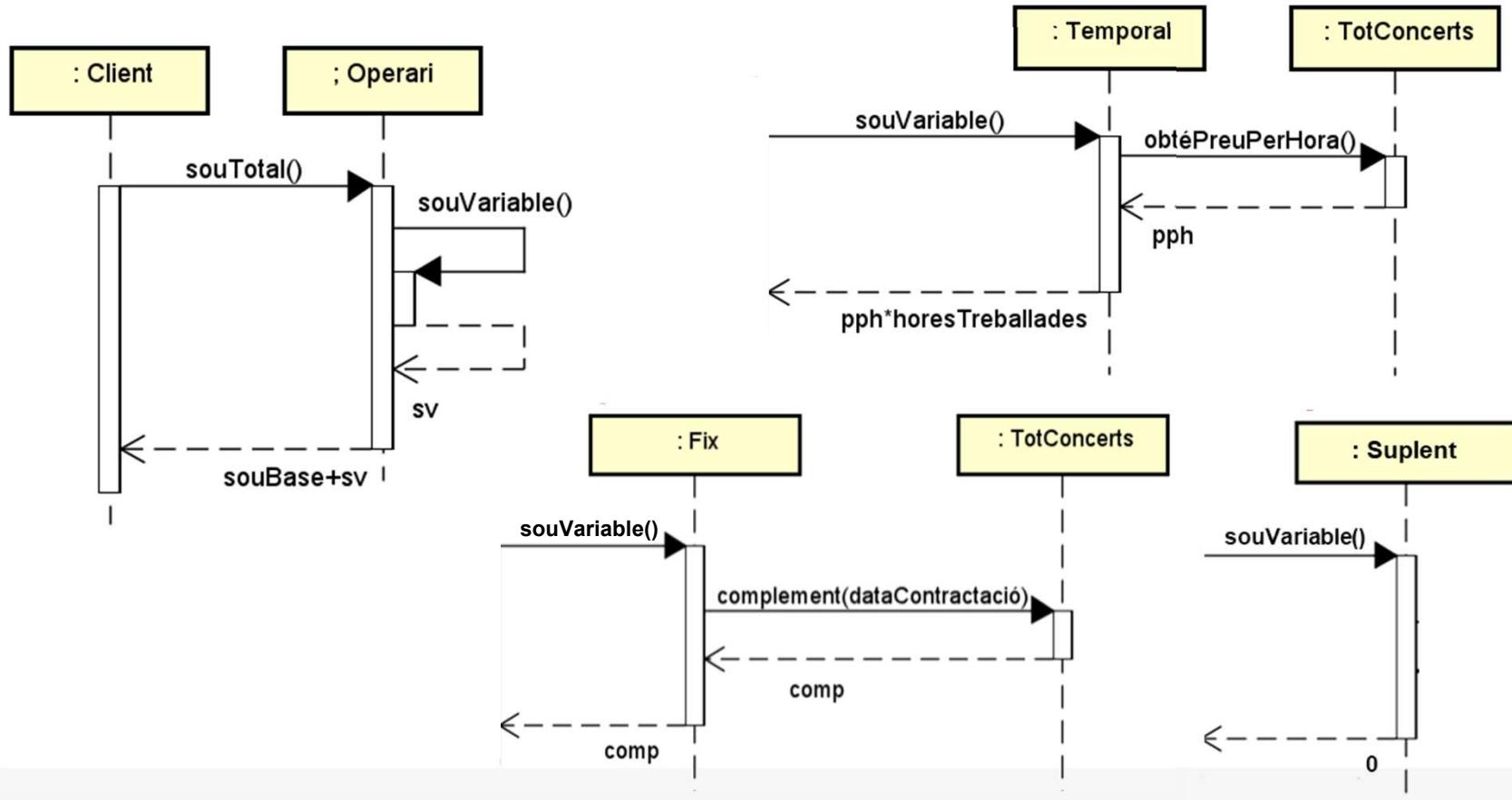
Patró Plantilla – exemple (revisitat)



Patró Plantilla – exemple, model de dades



Patró Plantilla – exemple, model de comportament



Referències

Bibliografia

- C. Larman. Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design. 3a ed. Prentice Hall, 2005.
- E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns. Addison-Wesley, 1995.
- R.C. Martin. Agile Software Development: Principles, Patterns and Practices. Prentice Hall, 2003.
- J. Rumbaugh, I. Jacobson, G. Booch. The Unified Modeling Language Reference Manual. 2a ed. Addison-Wesley, 2004.
- UML specification (v. 2.5.1). December 2017. <https://www.omg.org/spec/UML/>

Recursos

- <https://www.uml.org>