Principis de disseny de l'orientació a objectes

Principis de disseny de l'orientació a objectes

Disseny per contracte: excepcions als diagrames de seqüència
Operacions de la capa de dades
Polimorfisme i principi obert-tancat
Acoblament i cohesió
Bibliografia

Disseny per Contracte
Contractes de les operacions de disseny

Operació: nom i paràmetres de l'operació (*signatura de l'operació*)

Precondicions:
Condicions que estan garantides quan es crida l'operació.

Excepcions:
Condició que impedeix l'execució de l'operació quan no se satisfà.

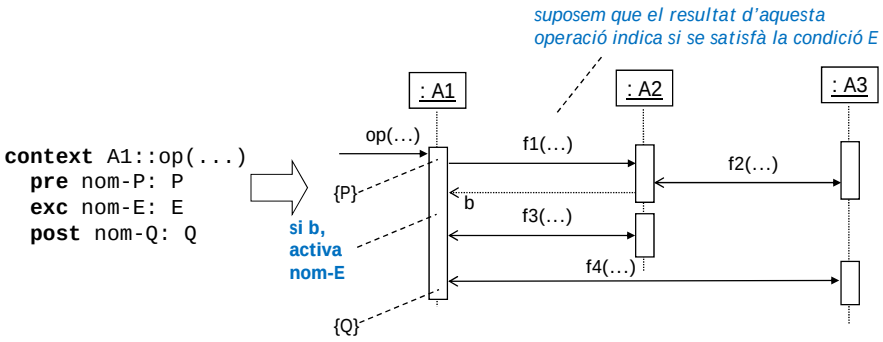
Postcondicions:
Canvis d'estat que es produeixen com a conseqüència de l'execució:

Sortida:
Descripció de la sortida que proporciona l'operació

Observeu que:
El concepte de precondició no és el mateix a especificació que a disseny
A disseny apareix un nou concepte, excepció, que captura el que a especificació era la precondició

Disseny per Contracte
Detecció i notificació de les excepcions als diagrames de seqüència

Quan una operació detecta una situació d'error declarada a l'apartat d'excepcions, diem que "activa" l'excepció

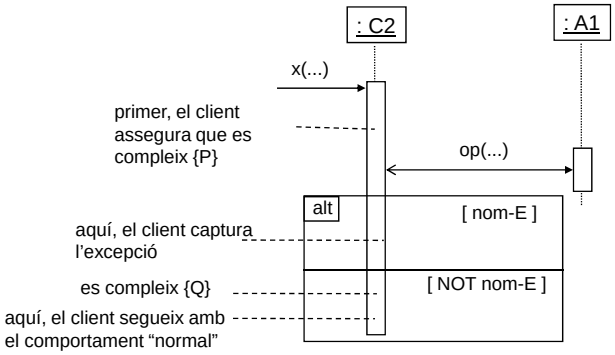


Convenció important: suposem que "activa" provoca que es desfacin els possibles canvis de l'estat del sistema que l'operació hagués efectuat

Disseny per Contracte

Captura de les excepcions als diagrames de seqüència

Quan un client rep notificació que s'ha produït una excepció, pot tractar-la si ho considera convenient




```
context C2::x(...)
  pre ...
  post ...
```

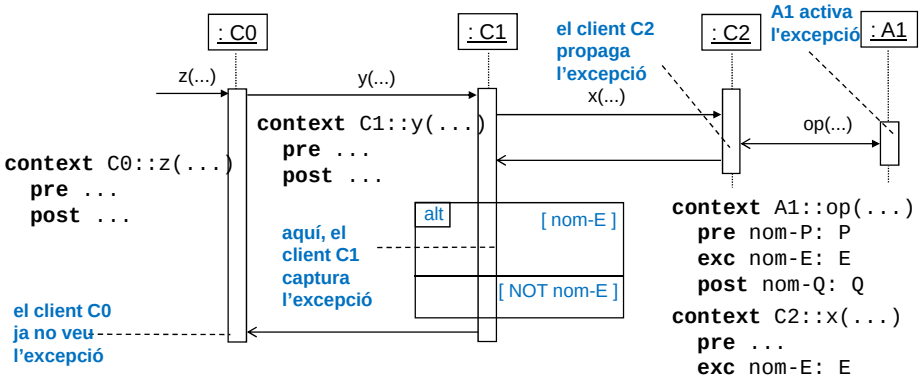
```
context A1::op(...)
  pre nom-P: P
  exc nom-E: E
  post nom-Q: Q
```


Disseny per Contracte

Propagació de les excepcions als diagrames de seqüència

Quan un client no vol capturar una possible excepció, simplement la propaga al seu propi client

- En algun punt de la cadena de crides, algun client ha de capturar-la
- Suposem que en propagar, també es desfan els possibles canvis efectuats en l'estat del sistema durant l'execució de l'operació




```
context C0::z(...)
  pre ...
  post ...
```

```
context C1::y(...)
  pre ...
  post ...
```

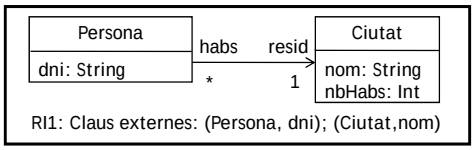
```
context A1::op(...)
  pre nom-P: P
  exc nom-E: E
  post nom-Q: Q
```

```
context C2::x(...)
  pre ...
  exc nom-E: E
  post ...
```

Operacions de la capa de dades

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

Les actualitzacions a la capa de dades són implícites



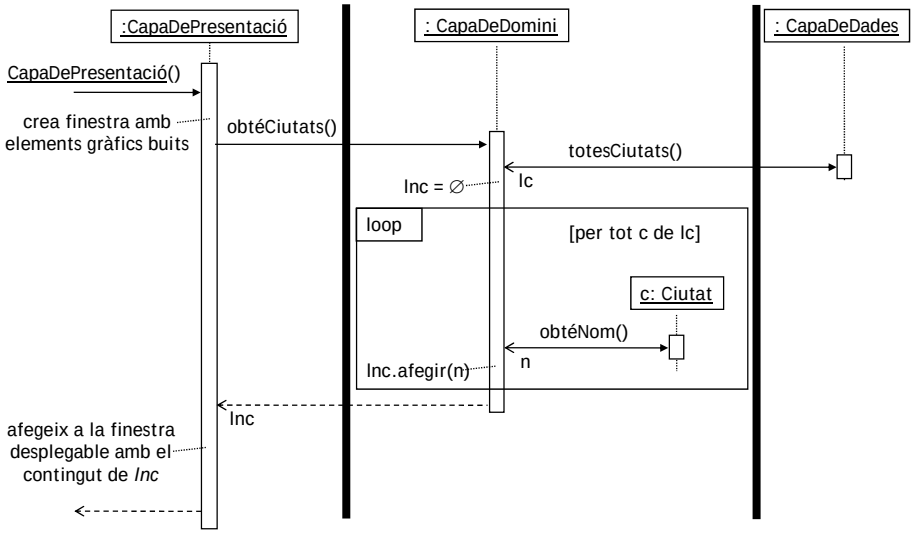
context CapaDeDades::totesCiutats(): Set(Ciutat)
post: 2.1 retorna les ciutats existents al sistema

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

context CapaDeDades::obtéCiutat(nom: String): Ciutat
exc ciutat-no-existeix**:** no existeix cap ciutat identificada per *nom*
post: 2.1 retorna ciutat amb *nom*

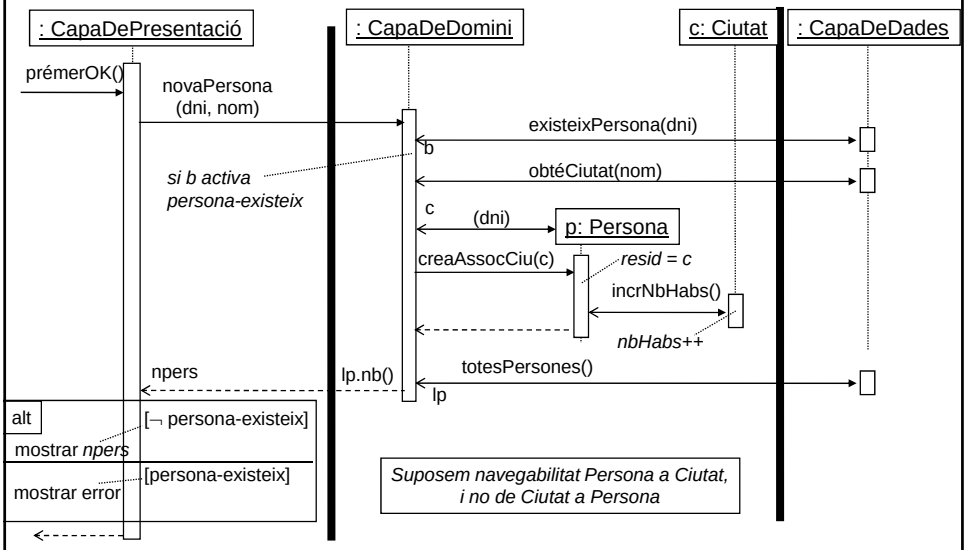
context CapaDeDades::existeixCiutat(nom: String): Bool
post: 2.1 retorna cert si existeix ciutat amb *nom*

Operacions de la capa de dades
Obtenció d'informació - exemple



Operacions de la capa de dades

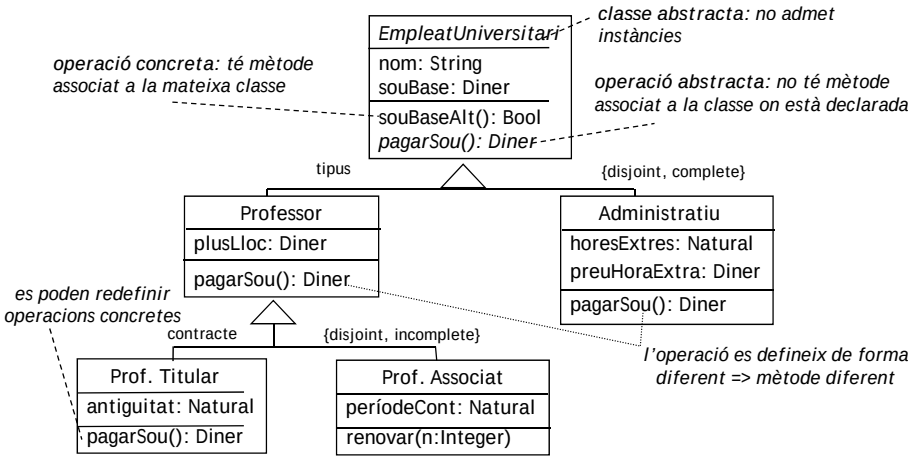
Alta d'informació - exemple



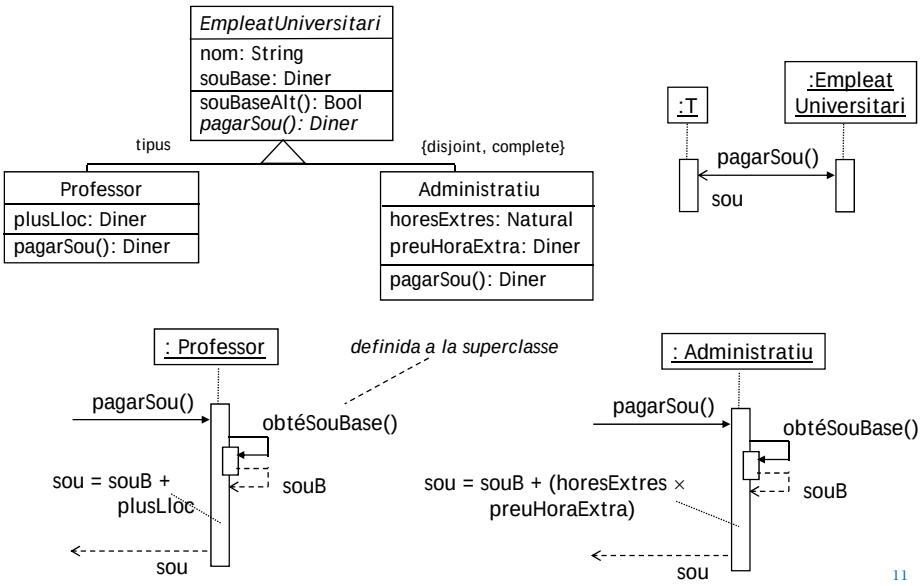
Polimorfisme

Operació polimòrfica:

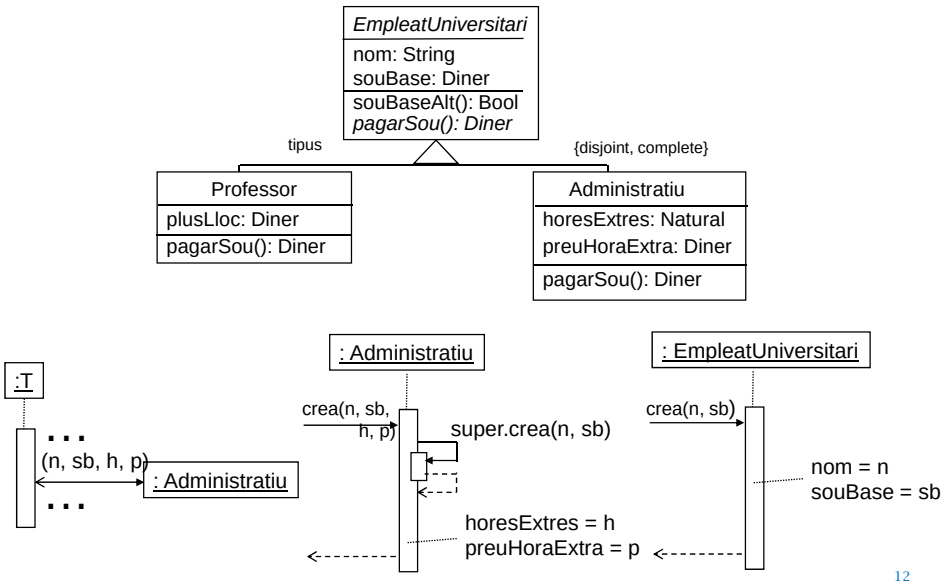
- operació que s'aplica a diverses classes d'una jerarquia tal que la seva semàntica depèn de la subclasse concreta on s'aplica



Polimorfisme: vinculació dinàmica



Creació d'objectes en una jerarquia d'herència



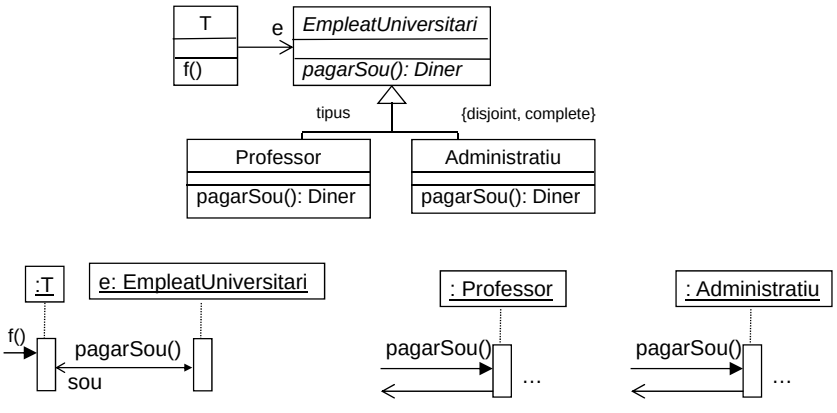
El principi Obert-Tancat (OCP)

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

El principio Obert-Tancat (OCP)

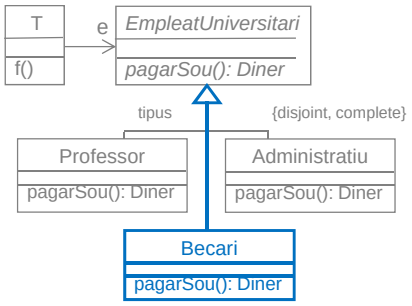
Satisfacció

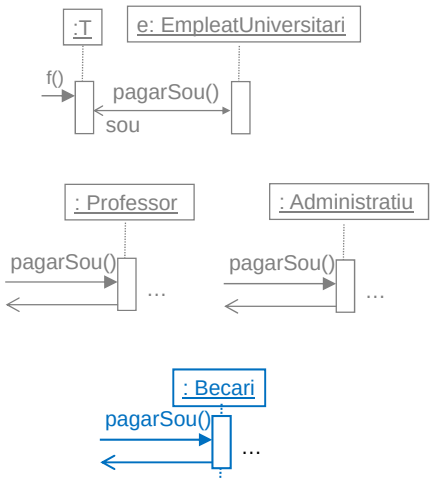
Totes les entitats software (classes, mòduls, funcions, etc.) haurien d'estar obertes per extensió, però tancades per modificació



El principio Obert-Tancat (OCP)

Nou tipus d'empleat: becari

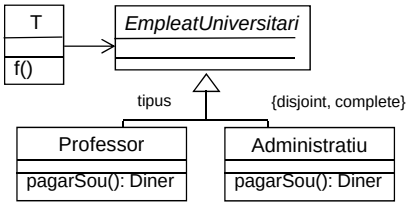


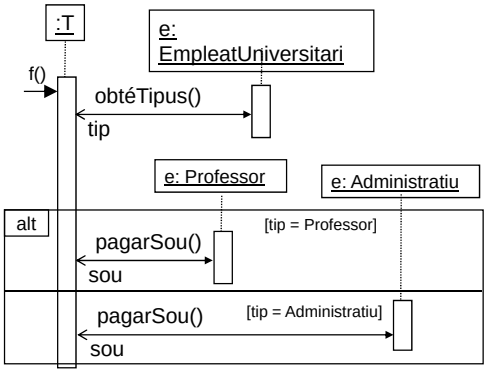


El principio Obert-Tancat (OCP)

Violació

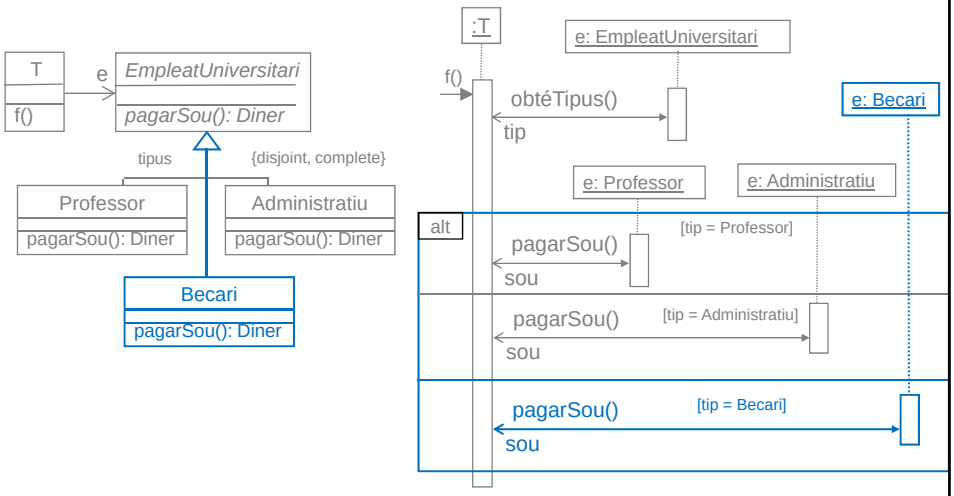
Totes les entitats software (classes, mòduls, funcions, etc.) haurien d'estar obertes per extensió, però tancades per modificació





El principio Obert-Tancat (OCP)

Nou tipus d'empleat: becari



Acoblament i cohesió

Regeixen la construcció d'arquitectures de qualitat

Hi ha diversos principis de disseny. Ens centrem en l'estudi de:

- l'acoblament → principi d'acoblament baix
- la cohesió → principi de cohesió alta

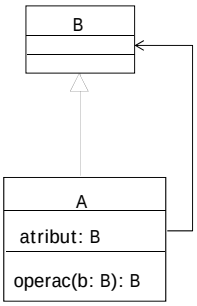
L'aplicació dels patrons de disseny es farà tenint en compte aquests dos principis

Acoblament

Acoblament d'una classe és una mesura del grau de connexió, coneixement i dependència d'aquesta classe respecte d'altres classes.

Per exemple, hi ha un acoblament de la classe A a la classe B si:

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



Principi de l'acoblament baix

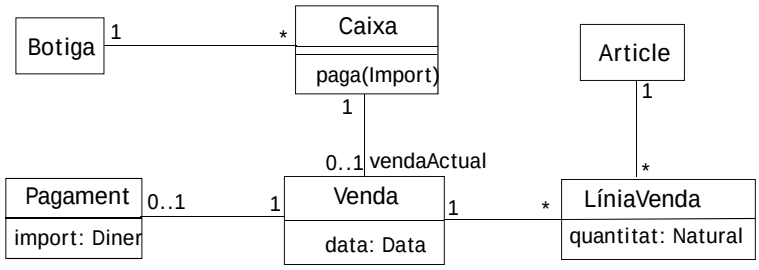
Convé que l'acoblament sigui baix:

- Si hi ha un acoblament de A a B, un canvi en B pot implicar canviar A.
- Quan més acoblament té una classe, més difícil resulta comprendre-la aïlladament.
- Quan més acoblament té una classe, és més difícil de reutilitzar-la, perquè requereix la presència de les altres classes.

Excepcions:

- L'acoblament amb classes estables ben conegudes no acostuma a ser problema (tipus de dades, classes de biblioteques ofertes pel llenguatge de programació, ...).

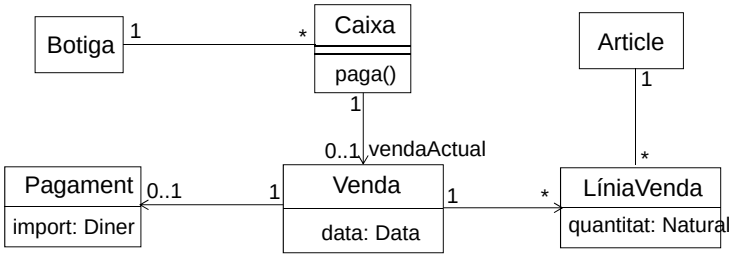
Acoblament
Exemple: diagrama de classes inicial i contracte a dissenyar




```
context Caixa::paga(import: Diner)
  pre: existeix vendaActual
  post: -- crea un pagament nou p amb p.import = import
        -- associa la vendaActual amb el pagament
```

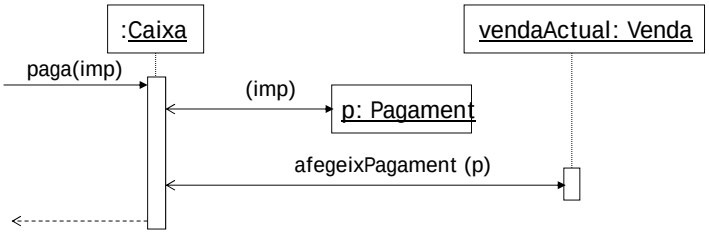

Acoblament
Exemple: diagrama de classes amb navegabilitat

Suposem un disseny que en un moment determinat presenta la navegabilitat següent i no té més acoblaments dels que es dedueixen del diagrama:



Acoblament
Exemple: disseny de l'operació *paga* (alternativa 1)

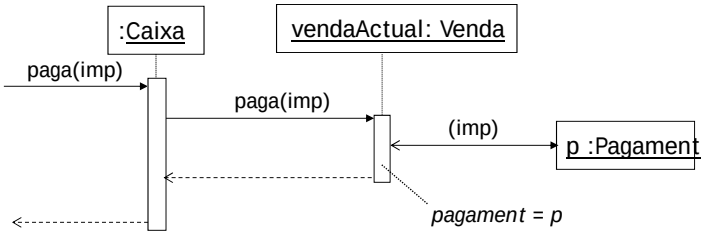
Alternativa 1: *Caixa* crea pagament i l'associa a *Venda*



Introdueix un nou acoblament (dinàmic) entre *Caixa* i *Pagament*

Acoblament
Exemple: disseny de l'operació *paga* (alternativa 2)

Alternativa 2: *Caixa* delega a *Venda* l'operació de creació del pagament



No introdueix cap nou acoblament

Llei de Demeter

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

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

Tots els altres objectes són "estranys". Per això, la llei també es coneix com a "No parleu amb estranys".

La llei de Demeter ajuda a mantenir l'acoblament baix

Cohesió

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

Convé que la cohesió sigui alta

Una classe amb cohesió alta:

- Té poques responsabilitats en una àrea funcional
- Col·labora (delega) amb d'altres classes per a fer les tasques
- Acostuma a tenir poques operacions. Aquestes operacions estan molt relacionades funcionalment

Avantatges:

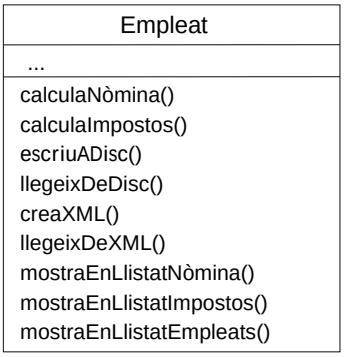
- Fàcil comprensió
- Fàcil reutilització i manteniment

No existeix una mètrica quantitativa simple de la cohesió

- Avaluació qualitativa

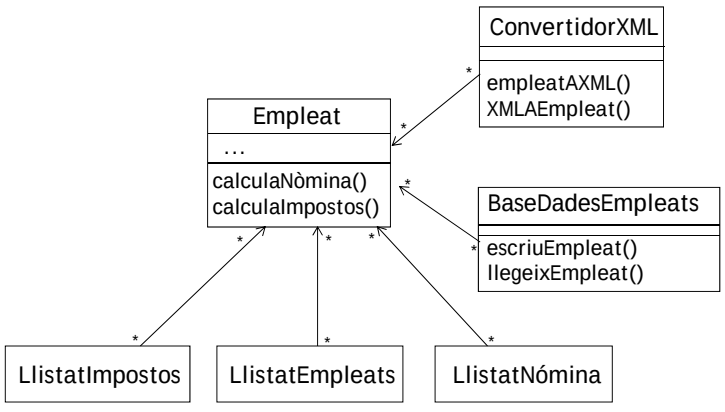
Cohesió: exemple (1)

Exemple de cohesió baixa:



Cohesió: exemple (2)

Exemple de cohesió alta:



Bibliografia

- Larman, C. "*Applying UML and Patterns. An Introduction to Object-oriented Analysis and Design*", Prentice Hall, 2005, (3ª edició).
- http://www.uml.org/#UML2.3
- Meyer, B. "*Object-Oriented Software Construction*", Prentice Hall, 1997, cap. 3
- Martin, R.C., "*Agile Software Development: Principles, Patterns and Practices* ", Prentice Hall, 2003, caps. 7 i 9.