

Patrons de disseny

Concepte de Patró

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”

Christopher Alexander, arquitecte (1977)

Patrons

- La Arquitectura en Capes forma part del que anomenarem un “Patró”.
- Un patró és una solució estandaritzada per a un problema comú que ens serveix de punt de partida per a atacar la situació actual.

Patrons

- Definició
- Patró Domain Model
- Patró Transaction Script
- Patró Controlador
 - Façana
 - Transacció
- Patró Expert

Concepte de patró

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”

Christopher Alexander, arquitecte (1977)

Context

- Situació en la què es presenta el problema de disseny

Problema

- Descripció del problema a resoldre
- Enumeració de les forces a equilibrar

Solució

- Aspecte estàtic: impacte en el diagrama de classes del disseny
- Aspecte dinàmic: establiment del comportament de les noves operacions

Catàlegs de patrons de disseny

Patrons que determinen l'estructura general de les capes

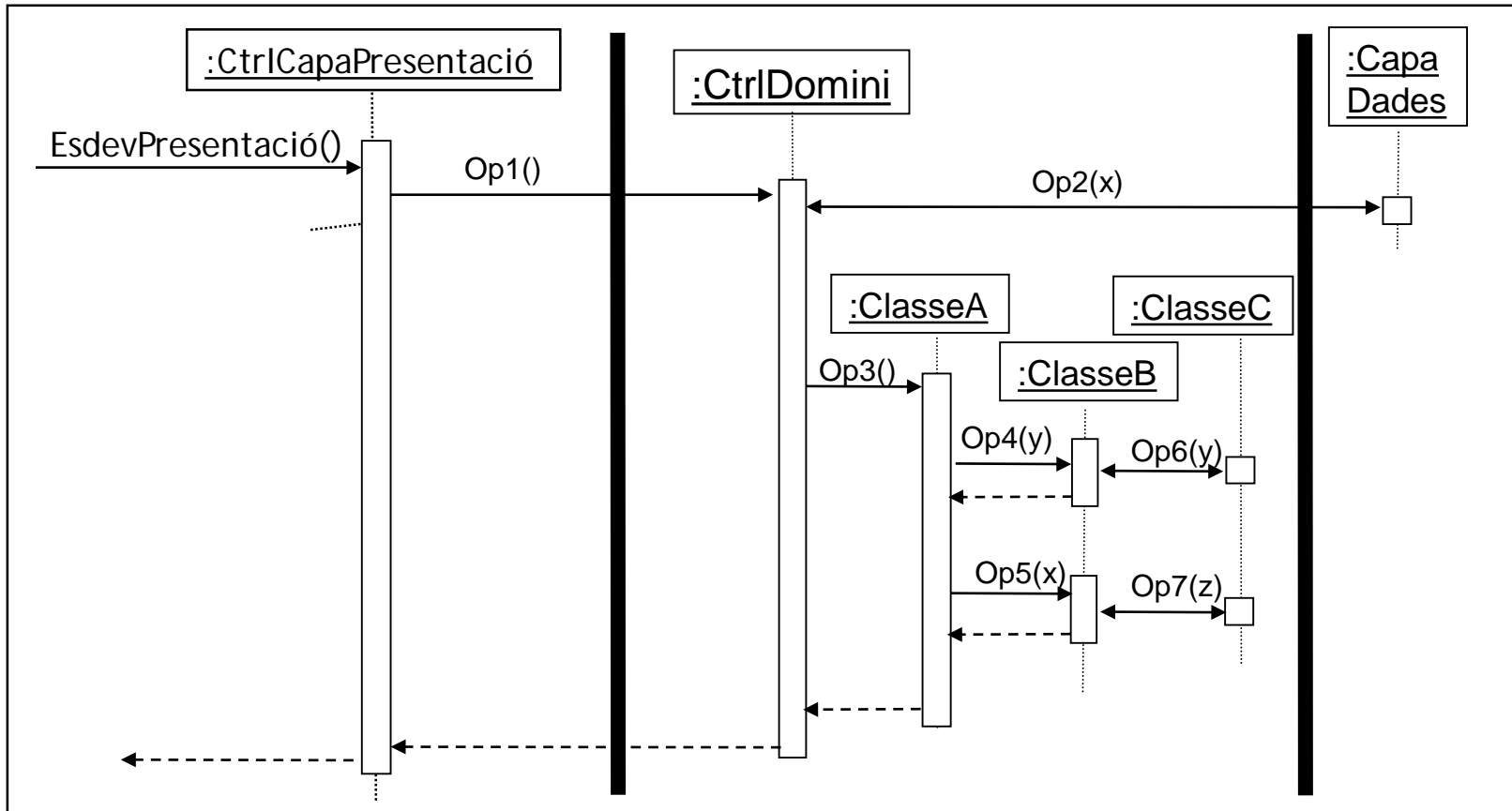
Proposats per Fowler (2003):

- Capa de domini:
 - Gran influència en la distribució de responsabilitats a capes
 - *Domain Model, Transaction Script*
- Capa de dades:
 - Determinen els serveis que ofereix la capa de dades
 - *Data Mapper, Pasarel·la Fila, Enregistrament Actiu*

Patró *Domain Model*

- La lògica de l'aplicació resideix bàsicament a la capa del domini
- La capa de domini implementa les seves operacions mitjançant la col·laboració d'instàncies de les seves classes:
 - Ús intensiu del concepte d'assignació de responsabilitats a nivell de classe
- Requereix:
 - Una transformació inicial de l'esquema conceptual d'especificació (dades i operacions) a un diagrama de classes i als contractes de les operacions de disseny
 - Conversió de la classe Data a atribut
- Característiques:
 - (+) Explota la riquesa pròpia de l'orientació a objectes
 - (+) Té a l'abast una col·lecció rica de patrons de disseny
 - (–) Pot no aprofitar-se completament de les funcionalitats ofertes pels SGBD

Domain Model: visió general d'un diagrama de seqüència



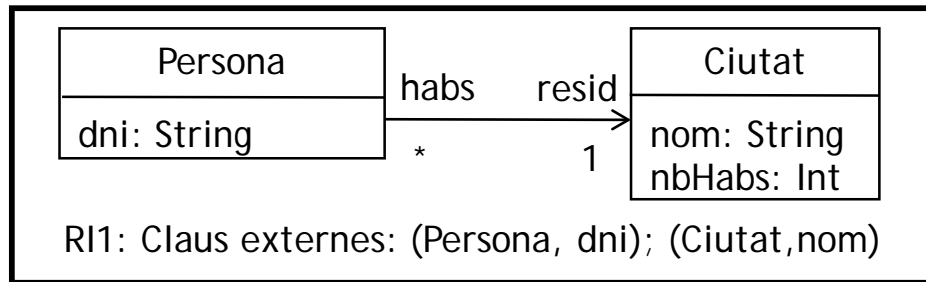
Resultat del disseny a IES:

- Diagrama de classes de disseny
- Contractes de disseny de les operacions
- Diagrama de seqüència de disseny per cada contracte

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
```

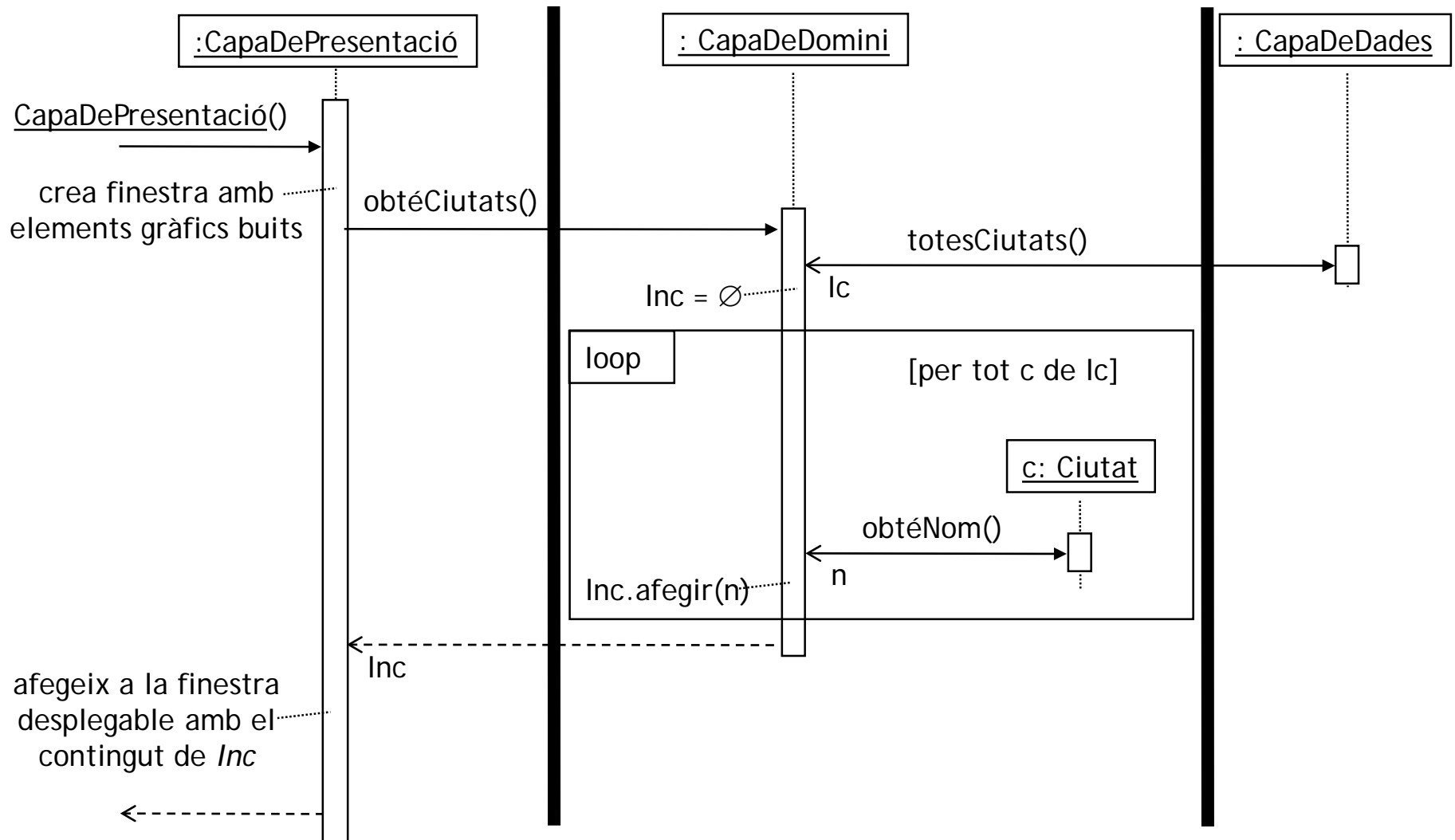
tot i que la controla el
SGBD, el mètode és qui
comunica l'error

```
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
```

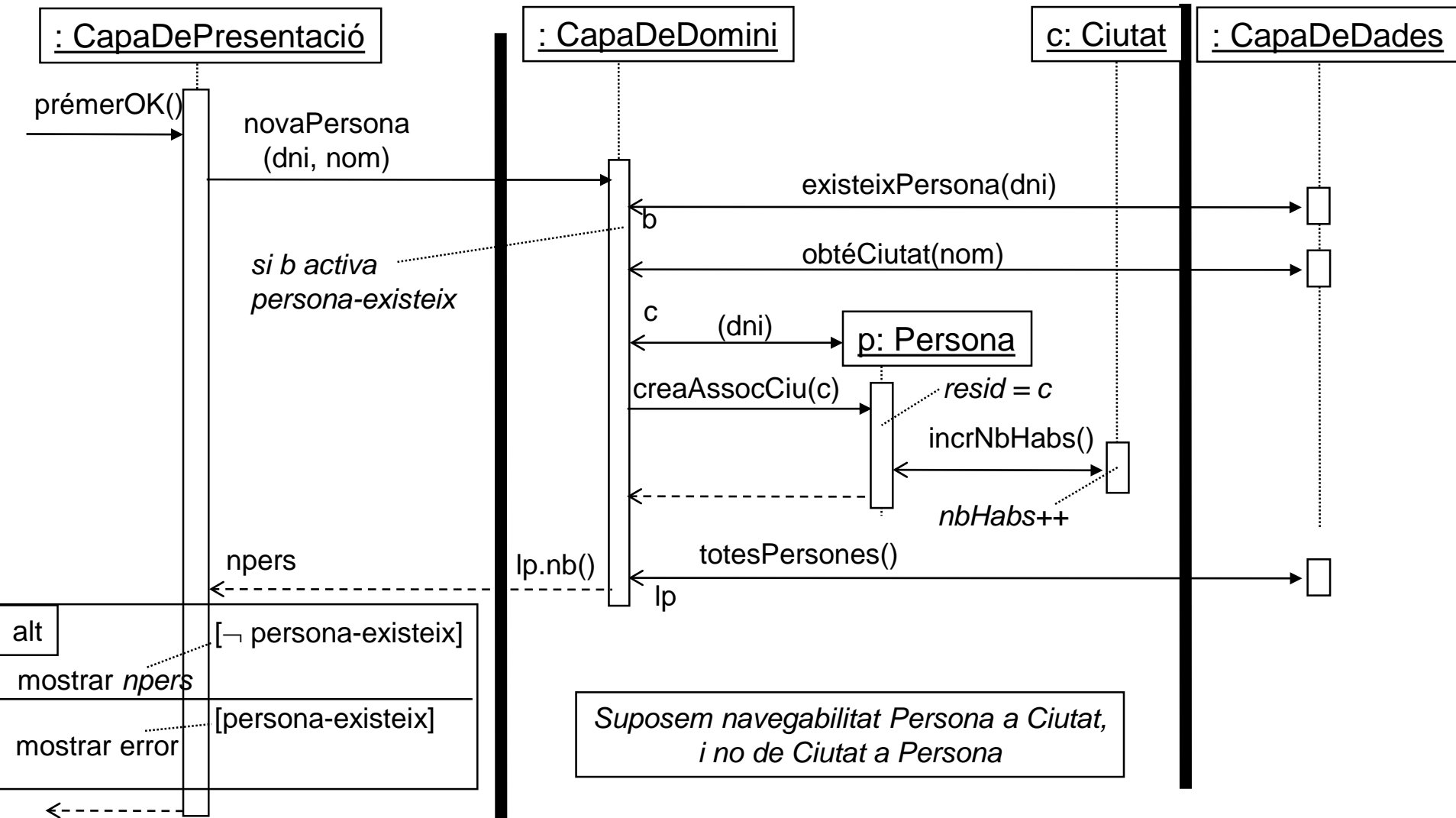
Operacions de la capa de dades

Obtenció d'informació - exemple



Operacions de la capa de dades

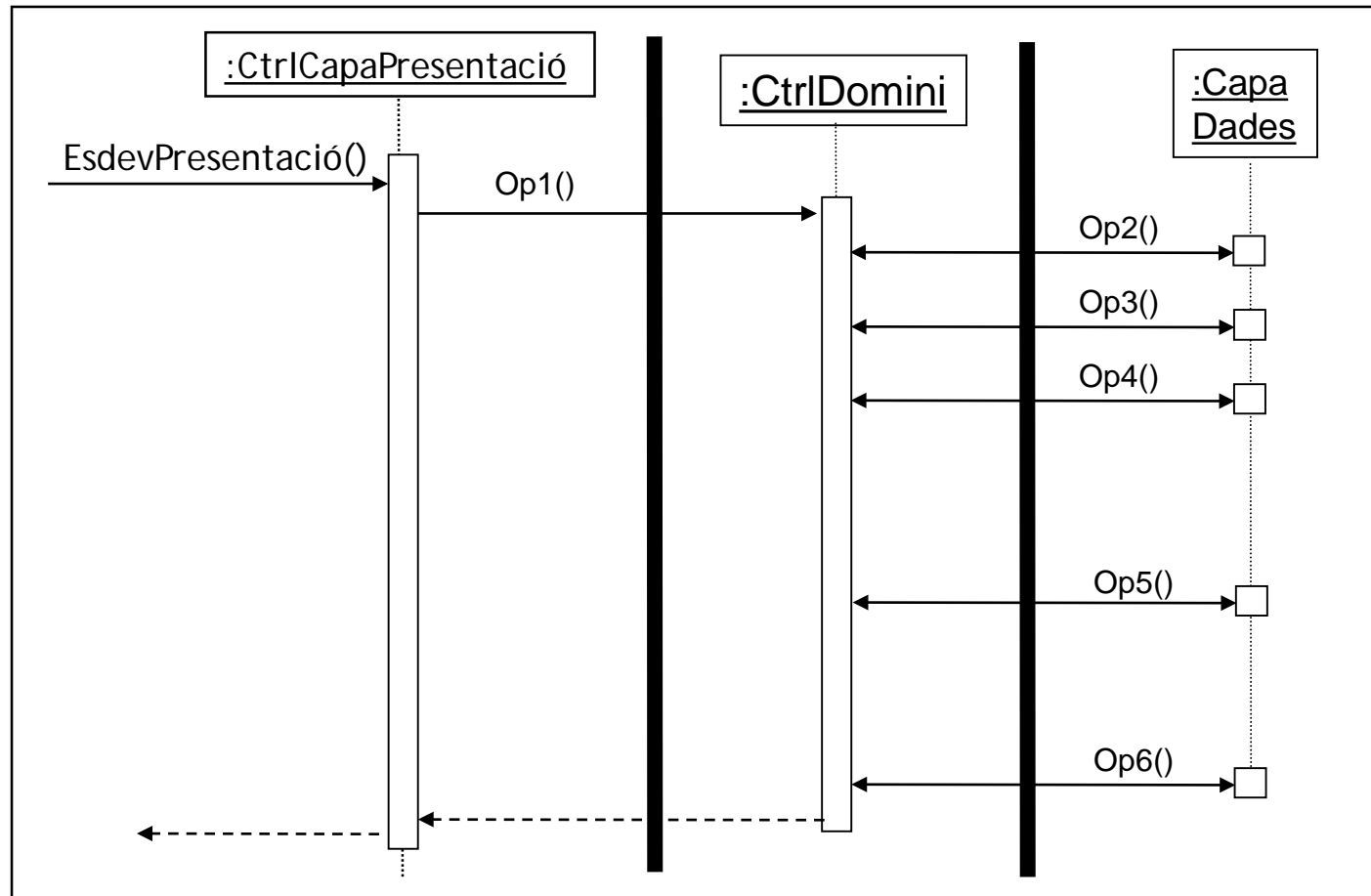
Alta d'informació - exemple



Patró *Transaction Script*

- Procediment que:
 - Rep les dades de la capa de presentació
 - Fa totes les validacions i càlculs necessaris
 - Es comunica amb la capa de dades per consultar i actualitzar la BD
 - Comunica els resultats a la capa de presentació
- Bàsicament, doncs, tenim un procediment per cada transacció de negoci
- La interacció amb la base de dades és totalment explícita
 - El disseny del software es fa considerant el SGBD que s'utilitzarà a la implementació
 - Serà diferent segons usem un SGBD orientat a objectes, relacional, etc.
- Característiques:
 - (+) Paradigma fàcil d'entendre pels programadors
 - (+) Capa de dades molt simple
 - (-) Solució complexa quan la lògica del domini creix
 - (-) La gestió de la persistència és explícita

Transaction Script: visió general d'un diagrama de seqüència



Patró controlador: descripció general

Context:

- Els (sub)sistemes software reben esdeveniments
 - . Ex: la capa de domini d'un SI rep esdeveniments externs
- Un cop interceptats aquests esdeveniments, algun objecte del sistema ha de rebre'ls i executar les accions corresponents

Problema:

- Quin objecte és el responsable de rebre un esdeveniment?

Solució:

- Assignar aquesta responsabilitat a un controlador
 - . Els clients del sistema desconeixen l'estructura interna del sistema
- Un controlador és un objecte d'una certa classe
 - . El controlador delega sobre un o més objectes del sistema el tractament de l'esdeveniment
- L'objecte que tracta l'esdeveniment no té coneixement sobre l'existència o el tipus de controlador
- Variants analitzades:
 - . Façana: Un objecte que representa tot el sistema
 - . Transacció (*Command*): Un objecte que representa una instància d'esdeveniment

Un exemple de Controlador: Facebook for Unity

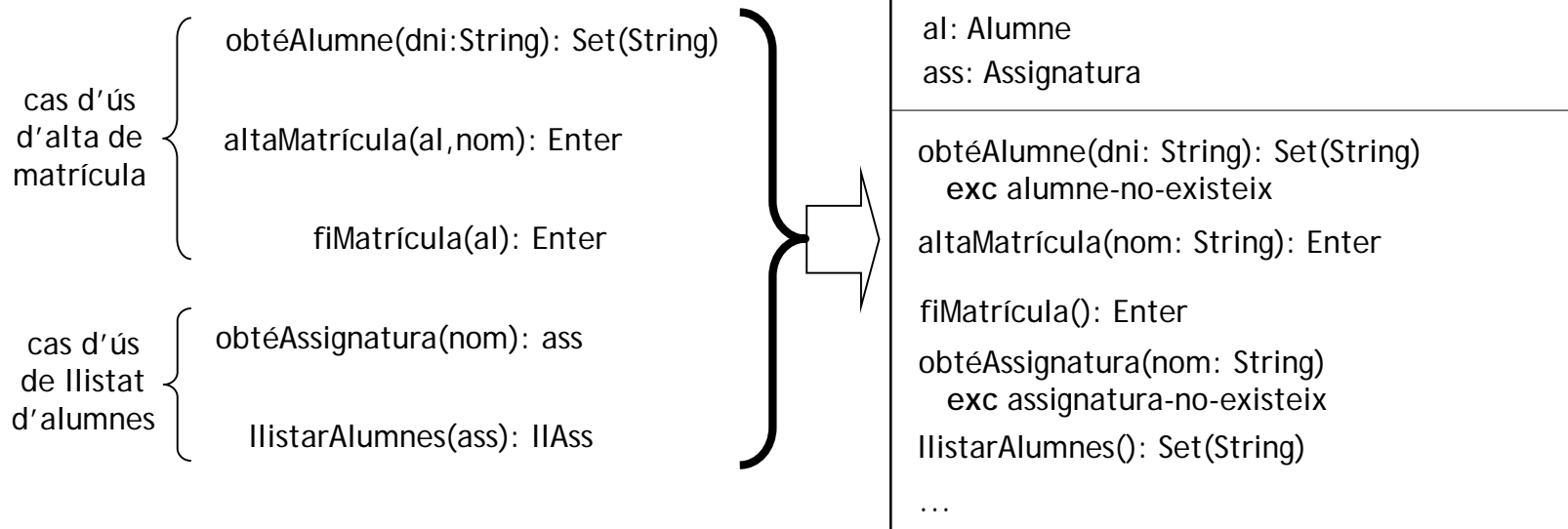
Name	Description
<code>FB.Init</code>	Initialize the SDK, this is required before doing anything else
<code>FB.API</code>	Make an API call to the Graph API
<code>FB.ShareLink</code>	Trigger a Share dialog for one-to-many sharing
<code>FB.FeedShare</code>	Trigger the legacy Feed sharing dialog, only use if you need legacy parameters
<code>FB.AppRequest</code>	Trigger a Game Request dialog for one-to-one sharing
<code>FB.GetAppLink</code>	Get the URL with which the app was invoked

Name	Description
<code>FB.LogInWithReadPermissions</code>	Prompt a user to authorize your app with requested read permissions, or to grant additional read permissions
<code>FB.LogInWithPublishPermissions</code>	Prompt a user to authorize your app with publish permissions, or to grant additional permissions
<code>FB.LogOut</code>	Log a user entirely out of Facebook

Controlador façana

Aspecte estàtic

- Classe *singleton*
 - tantes operacions com esdeveniments ha de capturar el sistema
 - eventualment, poden incloure's atributs per compartir informació
- Controladors inflats si hi ha molts esdeveniments → poca cohesió



Un exemple de Controlador: Facebook for Android

Nodes

Reading operations almost always begin with a **node**. A node is an individual object with a unique ID. For example, there are many User node objects, each with a unique ID representing a person on Facebook. To read a node, you query a specific object's ID. So, to read your User node you would query its ID:

[cURL](#)[Android SDK](#)[iOS SDK](#)[Java SDK](#)[PHP SDK](#)

```
GraphRequest request = GraphRequest.newMeRequest(  
    accessToken,  
    new GraphRequest.GraphJSONObjectCallback() {  
        @Override  
        public void onCompleted(JSONObject object, GraphResponse response) {  
            // Insert your code here  
        }  
    });
```

```
Bundle parameters = new Bundle();  
parameters.putString("fields", "id,name");  
request.setParameters(parameters);  
request.executeAsync();
```

Controlador transacció

Aspecte estàtic (1)

S'introdueix una classe concreta per cada operació del sistema (transacció)

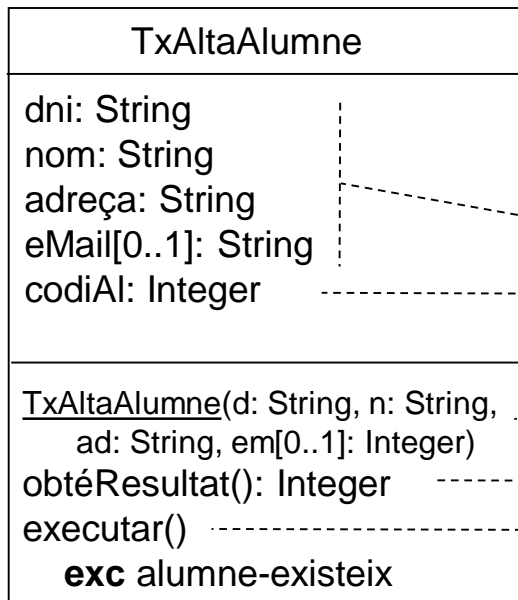
Cada paràmetre de l'operació dóna lloc a un atribut de la classe

- Si l'atribut és *out* o *inout*, s'afegeix una operació per consultar el seu valor
- L'operació constructora de la classe té tants paràmetres com paràmetres *in* i *inout* té l'operació

Si hi ha resultat, també es declara un atribut del tipus del resultat

- S'afegeix una operació per consultar el seu valor

S'afegeix una operació **executar()** que s'encarrega d'executar la transacció



Operació de sistema (transacció):

altaAlumne(dni: String, nom: String, adreça: String, eMail[0..1]: String):
Integer -- retorna el codi assignat a l'alumne

Paràmetres d'entrada de la transacció

Resultat de la transacció

Operació constructora amb els paràmetres adequats (cal declarar)

Consultora corresponent al resultat de la transacció

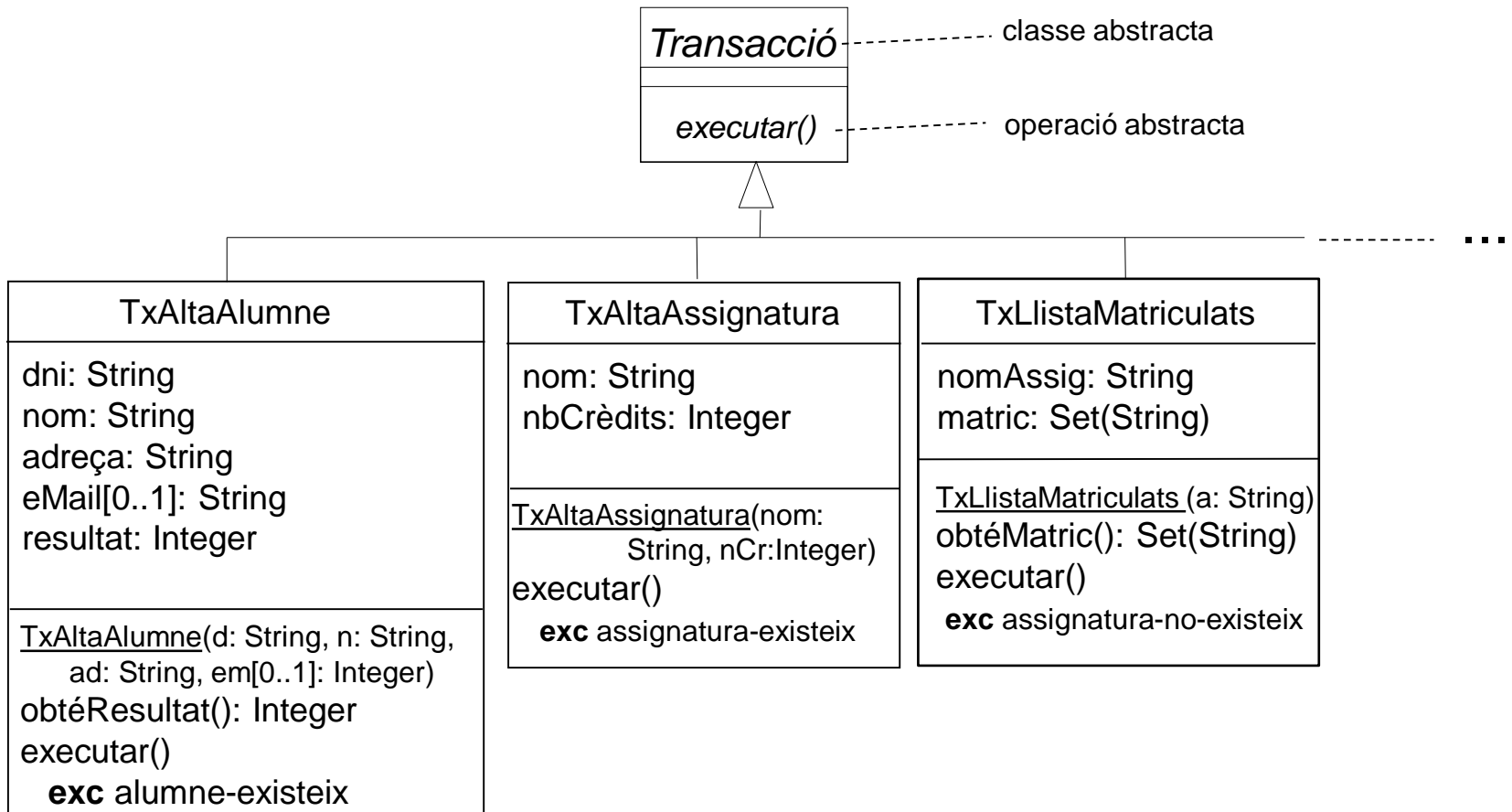
S'encarrega d'efectuar les accions corresponents a la transacció

Controlador transacció

Aspecte estàtic (2)

S'introdueix una classe abstracta que actua de superclasse de tots els controladors transacció del sistema

- Declara l'operació d'executar la transacció com a abstracta
- Proporciona una vista unificada a les classes clients dels diferents tipus de transaccions



Bibliografia

- Larman, C. *“Applying UML and Patterns. An Introduction to Object-oriented Analysis and Design”*, Prentice Hall, 2005, (3^a edició).
- Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. *“Pattern-oriented software architecture. A system of patterns”*, John Wiley & Sons, 1996.
- Fowler, M. *“Patterns of Enterprise Application Architecture”*, Addison-Wesley, 2002.
- Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *“Design Patterns”*, Addison-Wesley, 1995
- Martin, R.C., *“Agile Software Development: Principles, Patterns and Practices”*, Prentice Hall, 2003.