

Classe 11.10.2021: Introducció a Disseny

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,
Universitat de Barcelona

Curs 2021/22

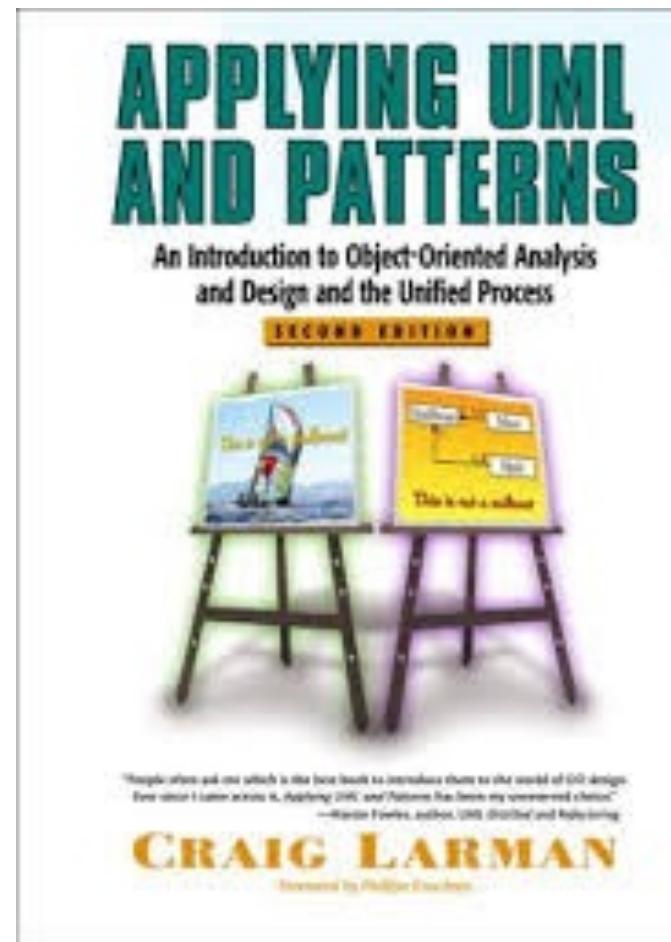


UNIVERSITAT DE
BARCELONA

Temari

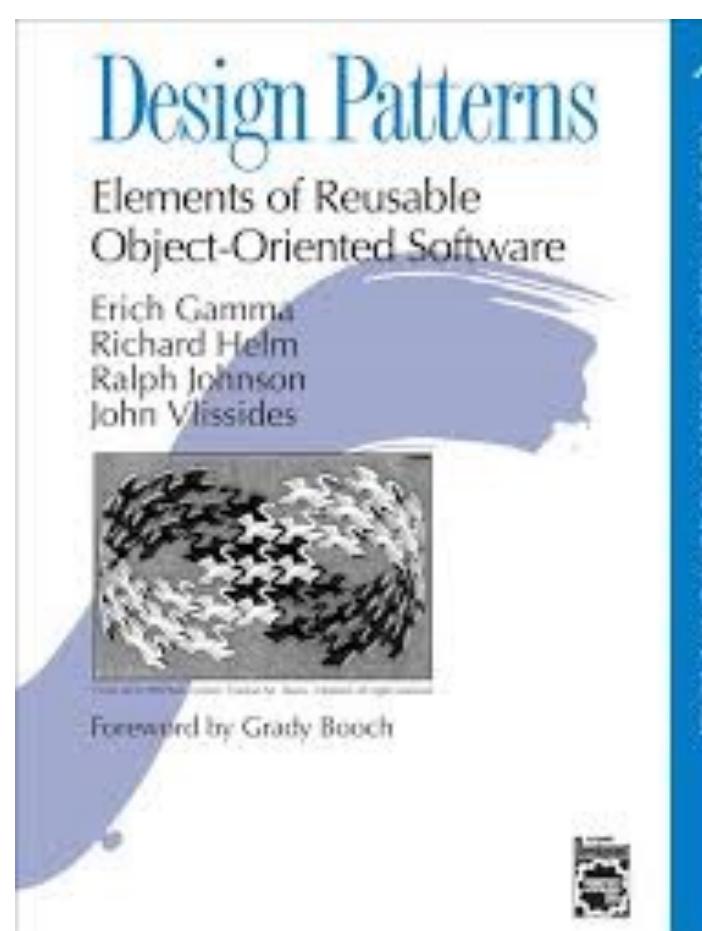
1	Introducció al procés de desenvolupament del software	
2	Anàlisi de requisits i especificació	
3	Disseny	3.1 Introducció
4	Del disseny a la implementació	3.2 Patrons arquitectònics
5	Ús de frameworks de testing	3.3 Criteris de Disseny: G.R.A.S.P. 3.4 Principis de Disseny: S.O.L.I.D. 3.5 Patrons de disseny

Introducció als patrons



Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process.

Molt generals però bones guies de disseny



Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

S'anomenen Gang of Four (GoF), descrit a la dècada dels 90 i descriu en detall 23 patrons de disseny

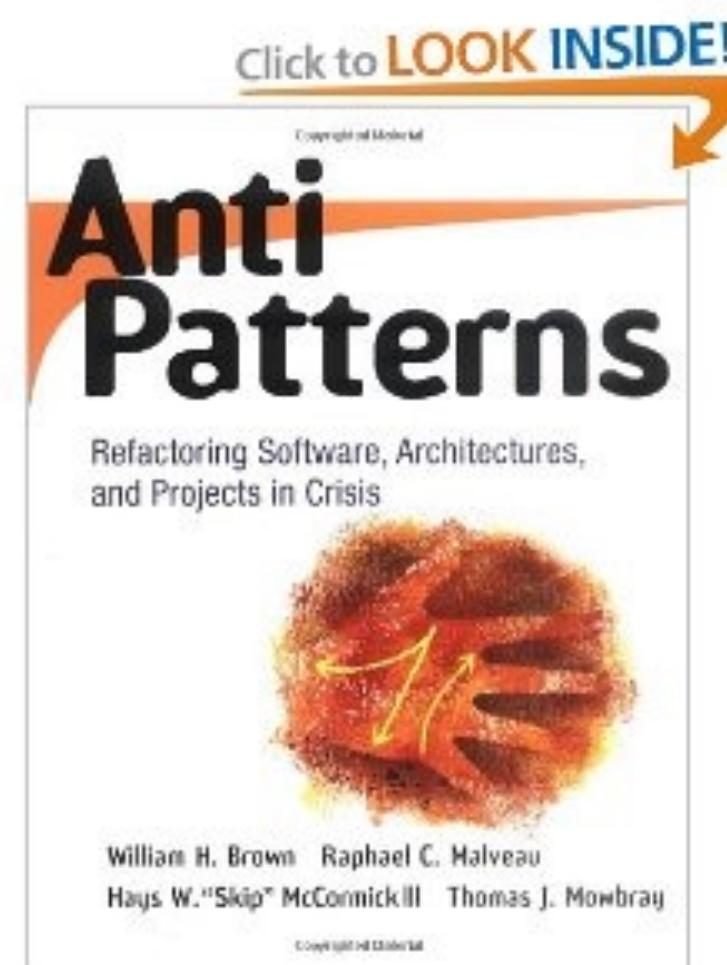
Molt utilitzats en les comunitats de desenvolupadors

Introducció als patrons



Mark Richards, Software Architecture Patterns, O'Reilly, 2015.

*Patrons arquitectònics bàsics amb exemples i
avaluacions*



William J. Brown, Raphael C. Malveau, Hays W. McCormick, Thomas J. Mowbray Wiley 1ra. Edició

*Si el concepte de **patrons** (bones solucions a problemes coneguts) resulta interessant, potser encara és més interessant el concepte d'**anti-patró** (errors comuns solucionant problemes coneguts)*

Temari (vídeos)



Introducció al disseny (metodologia i patrons)



Exemple de la Metodologia a seguir

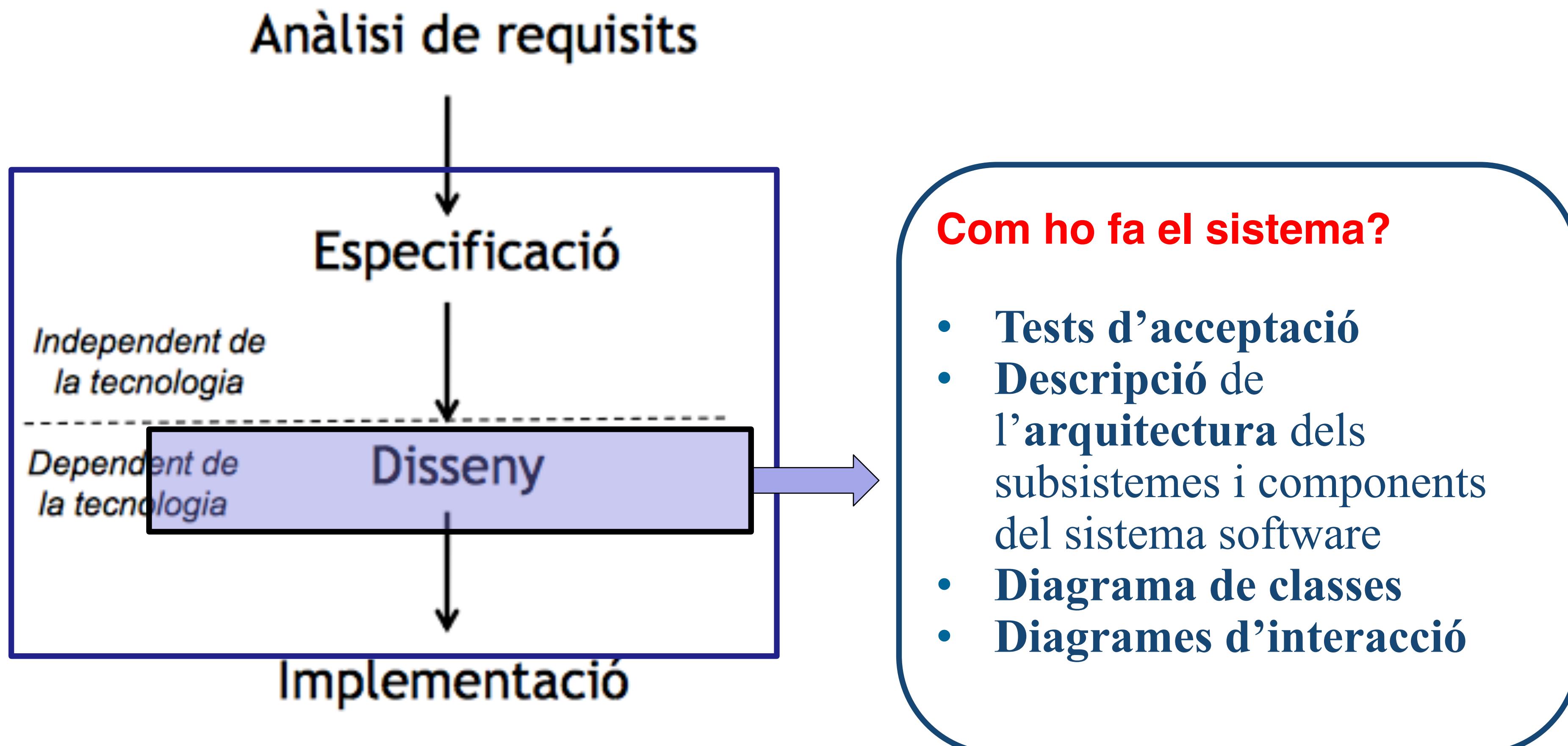


Propietats del paradigma Orientat a Objectes

Transparències de suport Tema 3.1

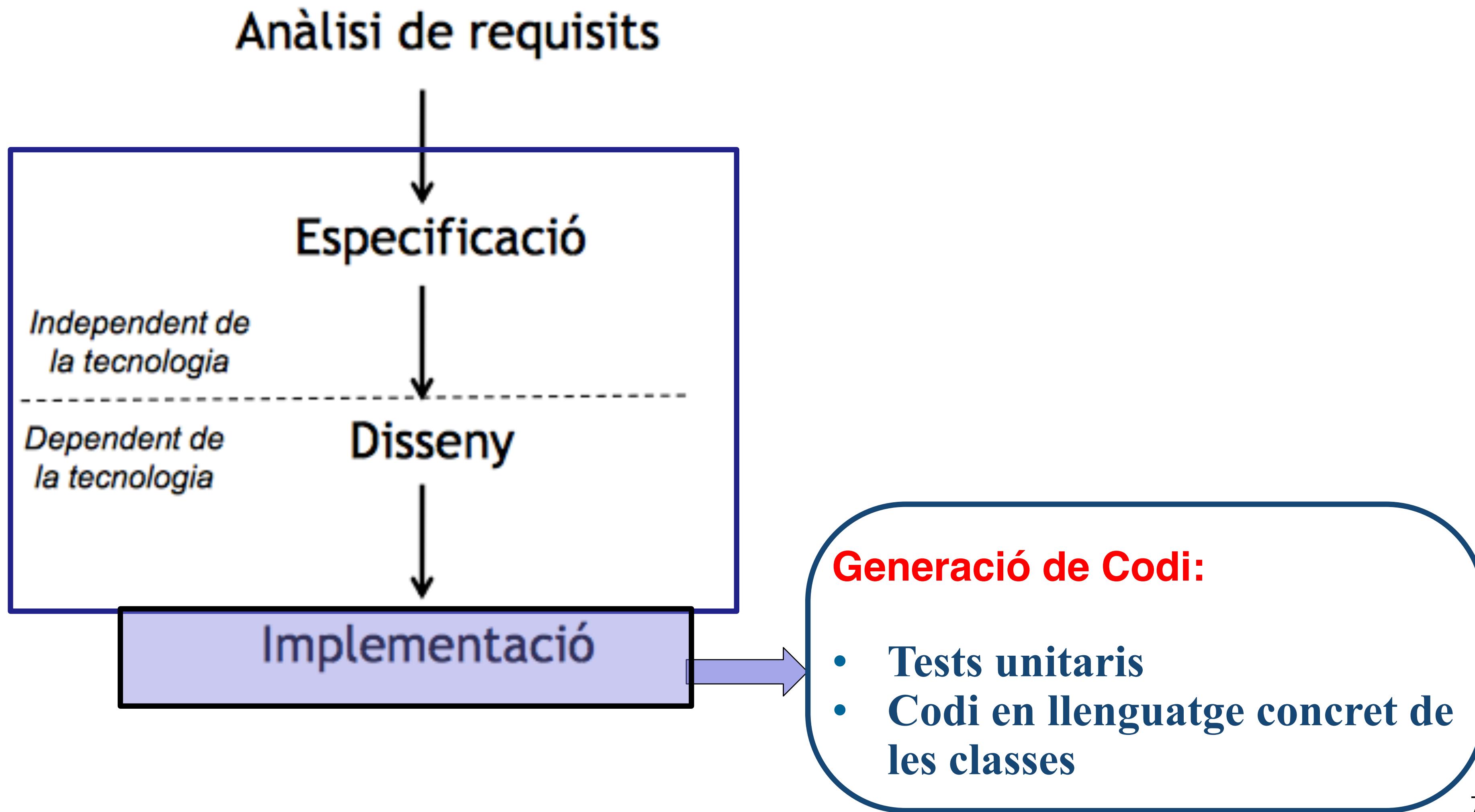
3.1. Introducció

Procés sistemàtic (Disseny):



3.1. Introducció

Procés sistemàtic (Desenvolupament):

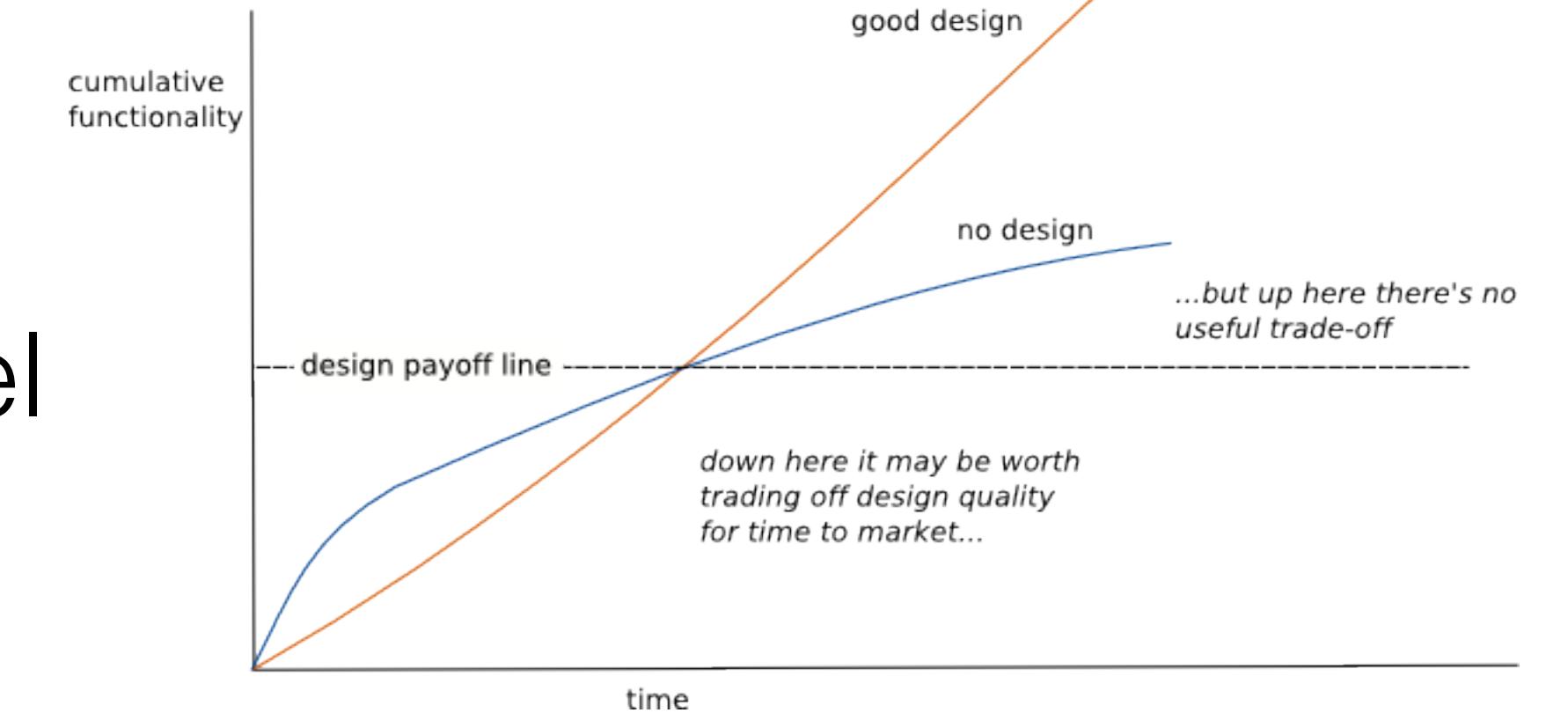


3.1. Introducció

Per a què es vol un **bon** disseny de software?

- Per manegar de forma **fiable** la **complexitat** del problema
- Per a **desenvolupar ràpid** i lliurar-lo a temps
- Per poder incloure **canvis** fàcilment

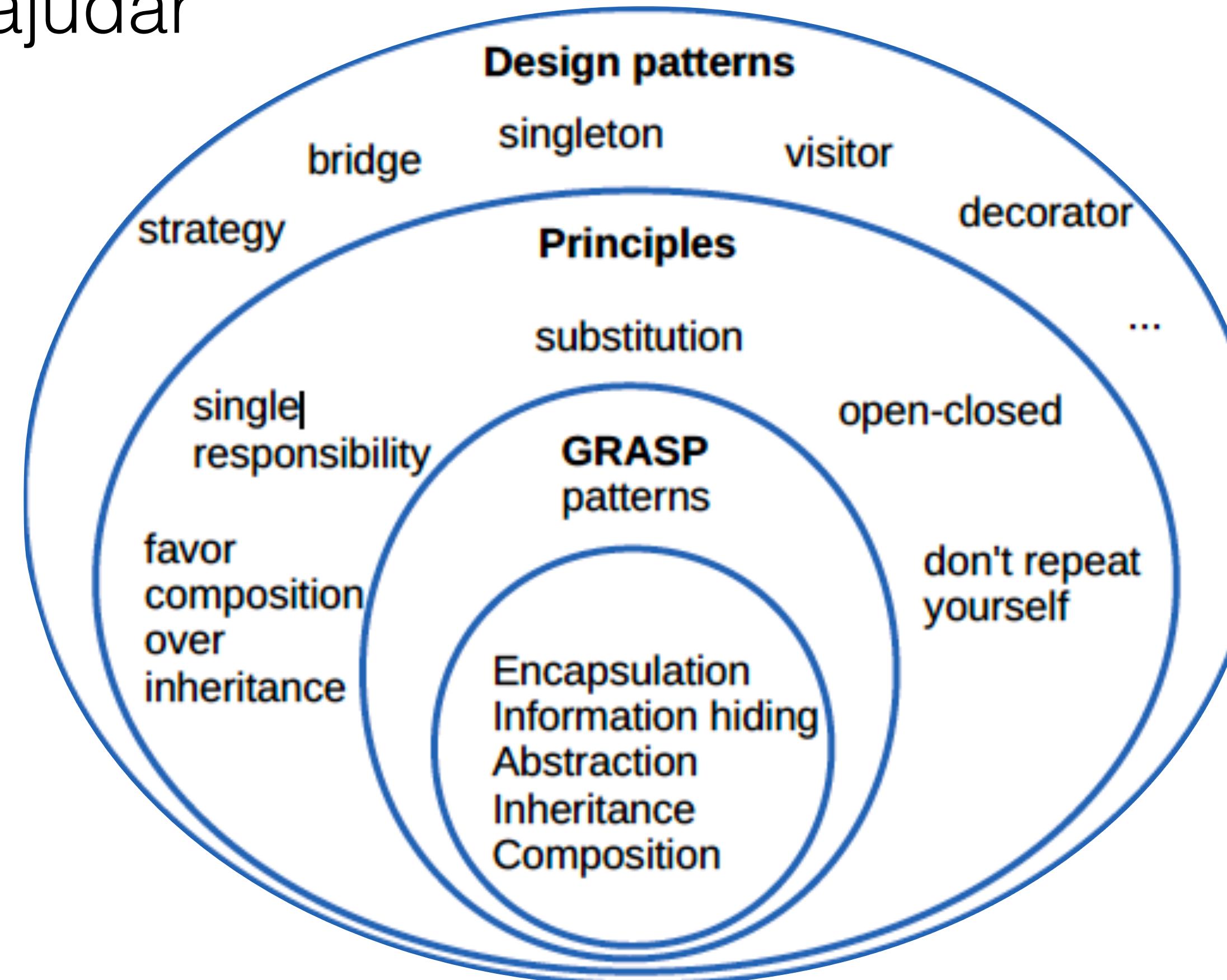
- Preservar els principis de disseny
- Adaptació de solucions genèriques a problemes coneguts de disseny (**patrons**)



<https://campusvirtual.ub.edu/mod/url/view.php?id=2668526>

Com disseny en OO?

- No hi ha una metodologia que doni el millor disseny però hi han principis, heurístiques i patrons que poden ajudar



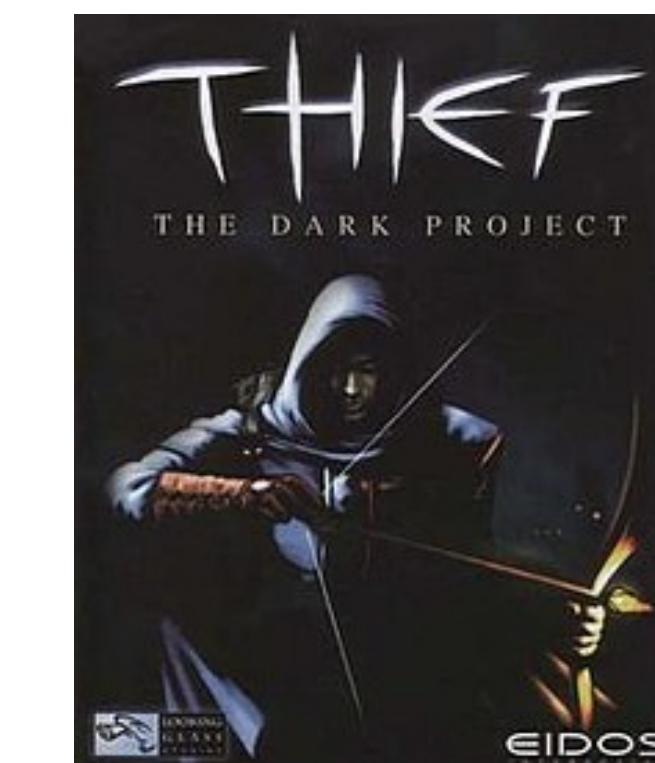
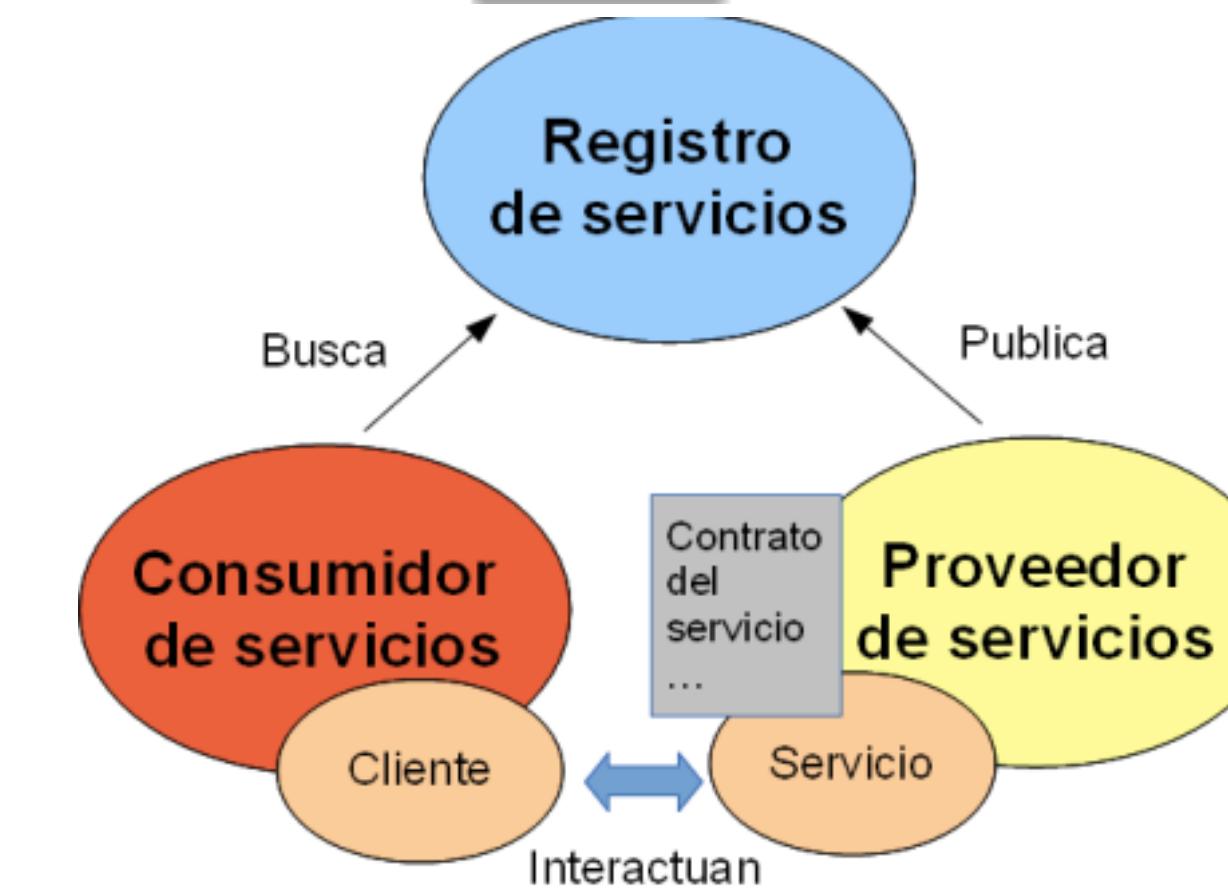
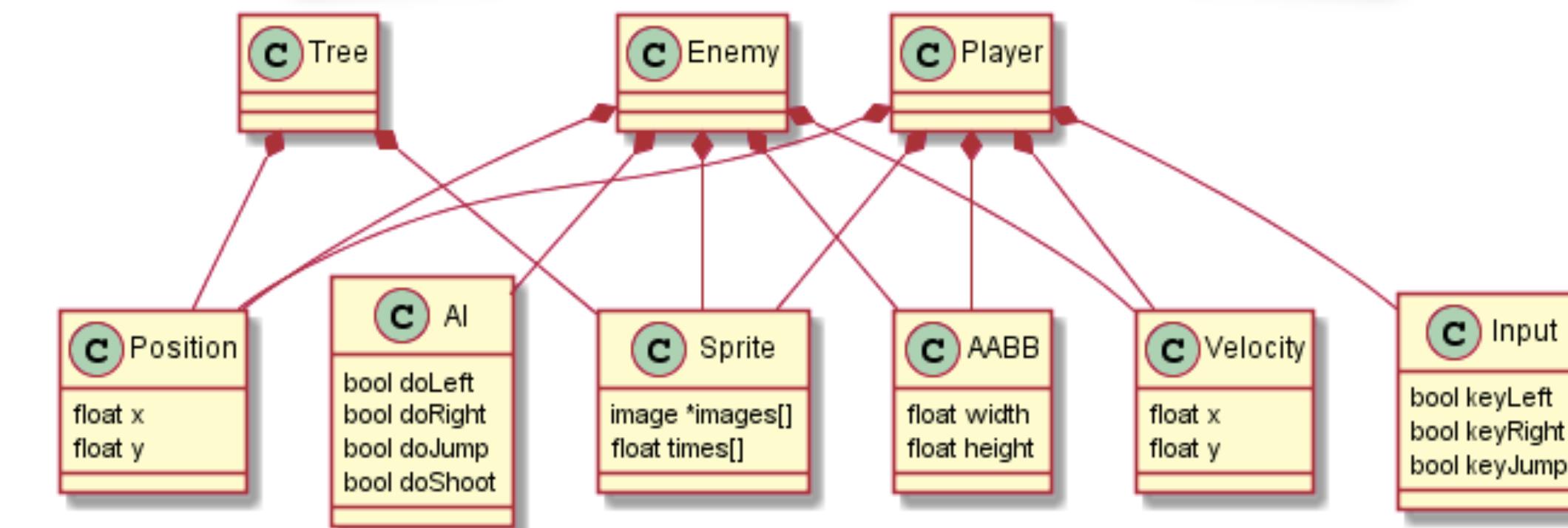
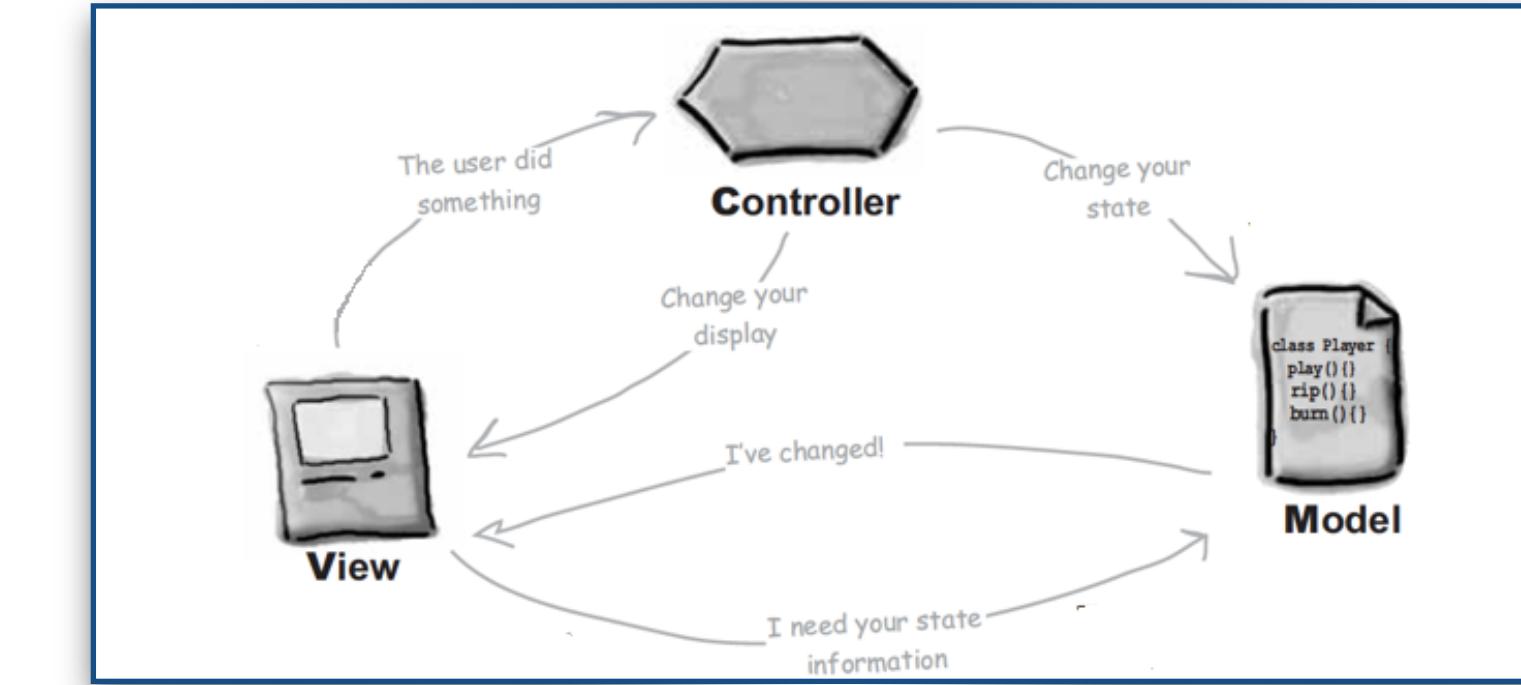
3.1. Introducció

- **Patrons d'arquitectura:** usats en el disseny a gran escala i de gra guixut.
 - *Per exemple:* patró de **Capes**
- **Patrons de disseny:** utilitzats en el disseny d'objectes i frameworks de petita i mitjana escala. (micro-arquitectura)
 - *Per exemple,* patró **Façana** (*Facade*) per connectar les capes o el patró **Estratègia** per permetre algorismes connectables
- **Patrons d'estils:** solucions de baix nivell orientades a la implementació o al llenguatge.
 - *Per exemple,* patró **Singleton** per fer una única instància d'una classe.

3.1. Introducció

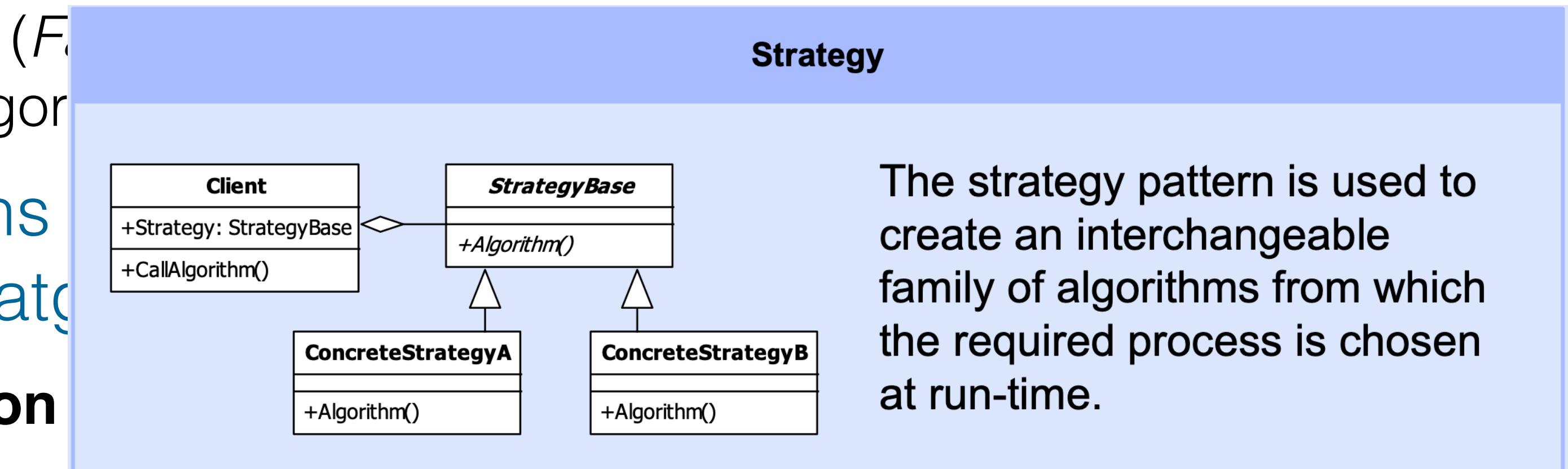
Patrons arquitectònics
(entre d'altres):

- Model-Vista-Controller
- Entity-Component System
- Service Oriented Architecture (SOA)



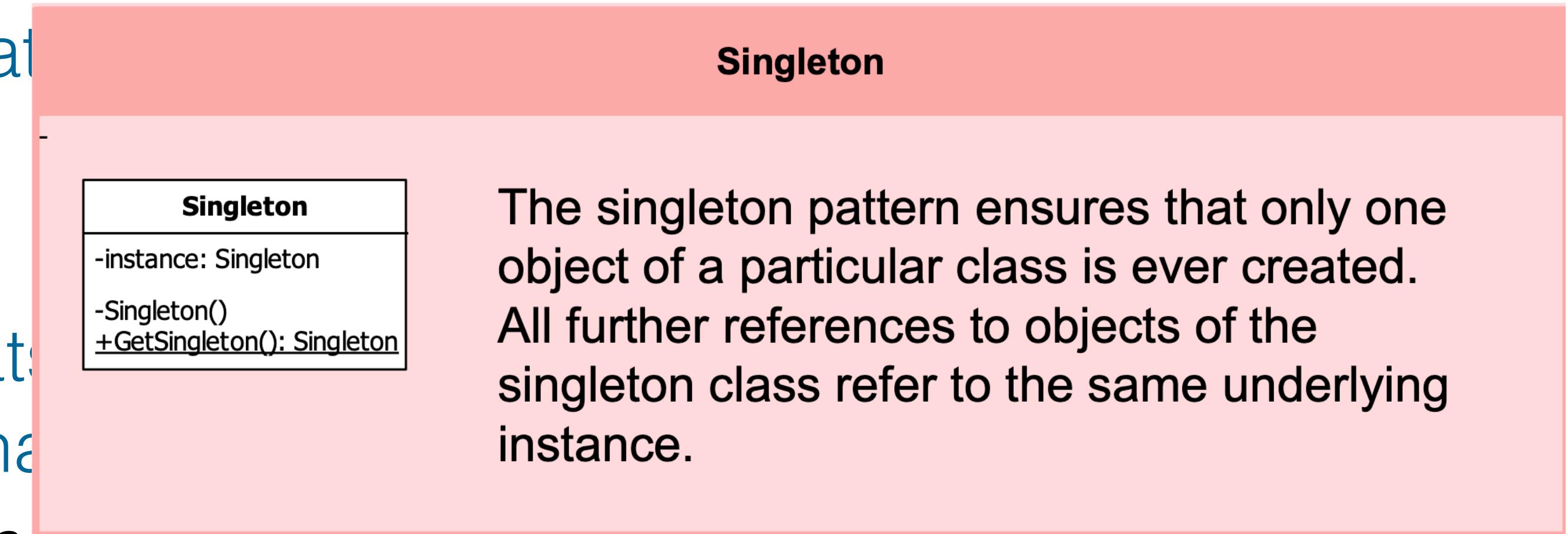
3.1. Introducció

- **Patrons d'arquitectura:** usats en el disseny a gran escala i de gra guixut.
 - *Per exemple:* patró de **Capes**
- **Patrons de disseny:** utilitzats en el disseny d'objectes i frameworks de petita i mitjana escala. (micro-arquitectura)
 - *Per exemple,* patró **Façana** (*Facade*) o **Estratègia** per permetre algorismes interchangeables.
- **Patrons d'estils:** solucions implementació o al llenguatge
 - *Per exemple,* patró **Singleton**



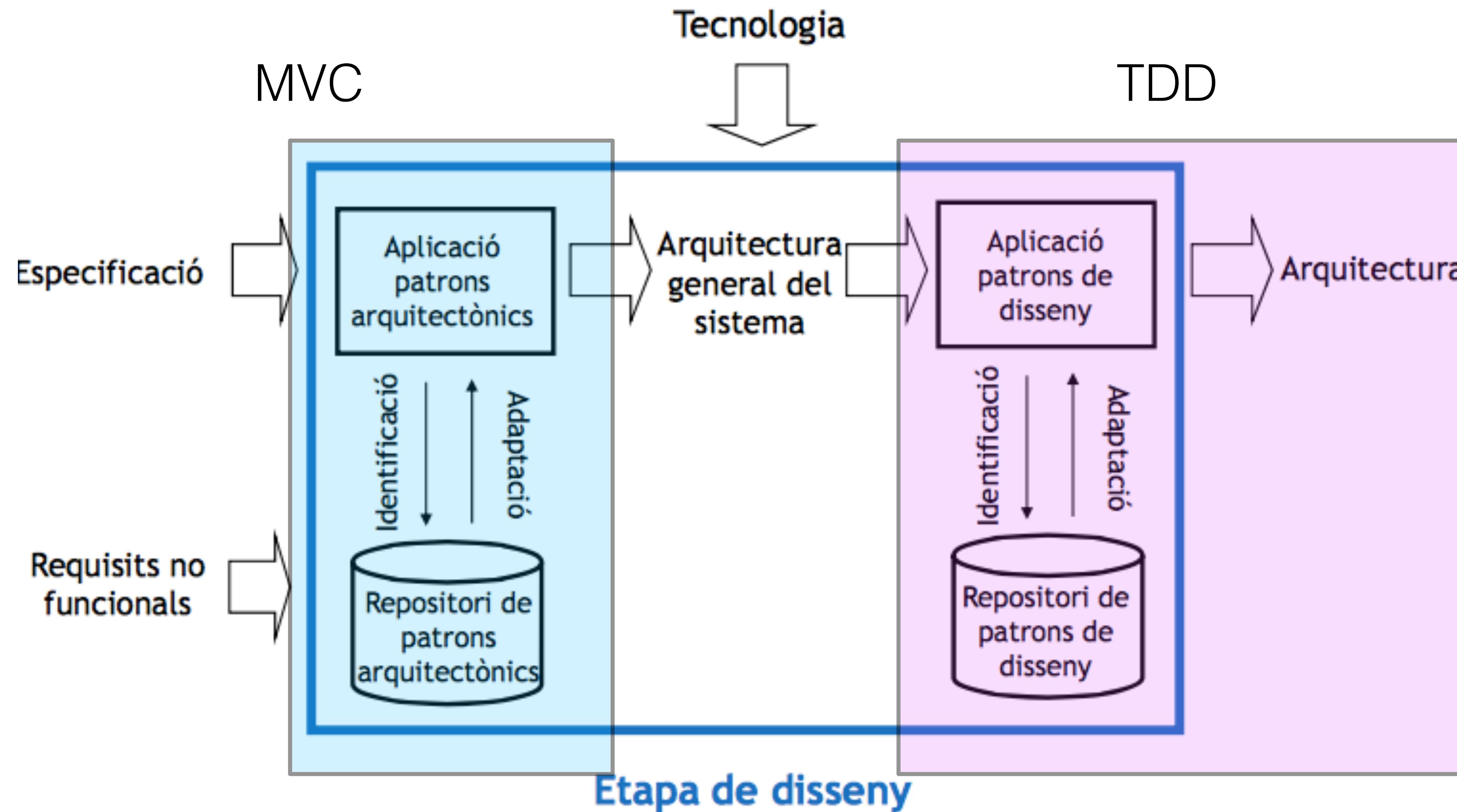
3.1. Introducció

- **Patrons d'arquitectura:** usats per a la creació d'una estructura de programació.
- *Per exemple:* patró de **Capes** per a la creació d'una arquitectura de programació.
- **Patrons de disseny:** utilitzats per a la creació d'una estructura de programació.
- *Per exemple,* patró **Façana** (*Facade*) per connectar les capes o el patró **Estratègia** per permetre algorismes connectables
- **Patrons d'estils:** solucions de baix nivell orientades a la implementació o al llenguatge.
- *Per exemple,* patró **Singleton** per fer una única instància d'una classe.



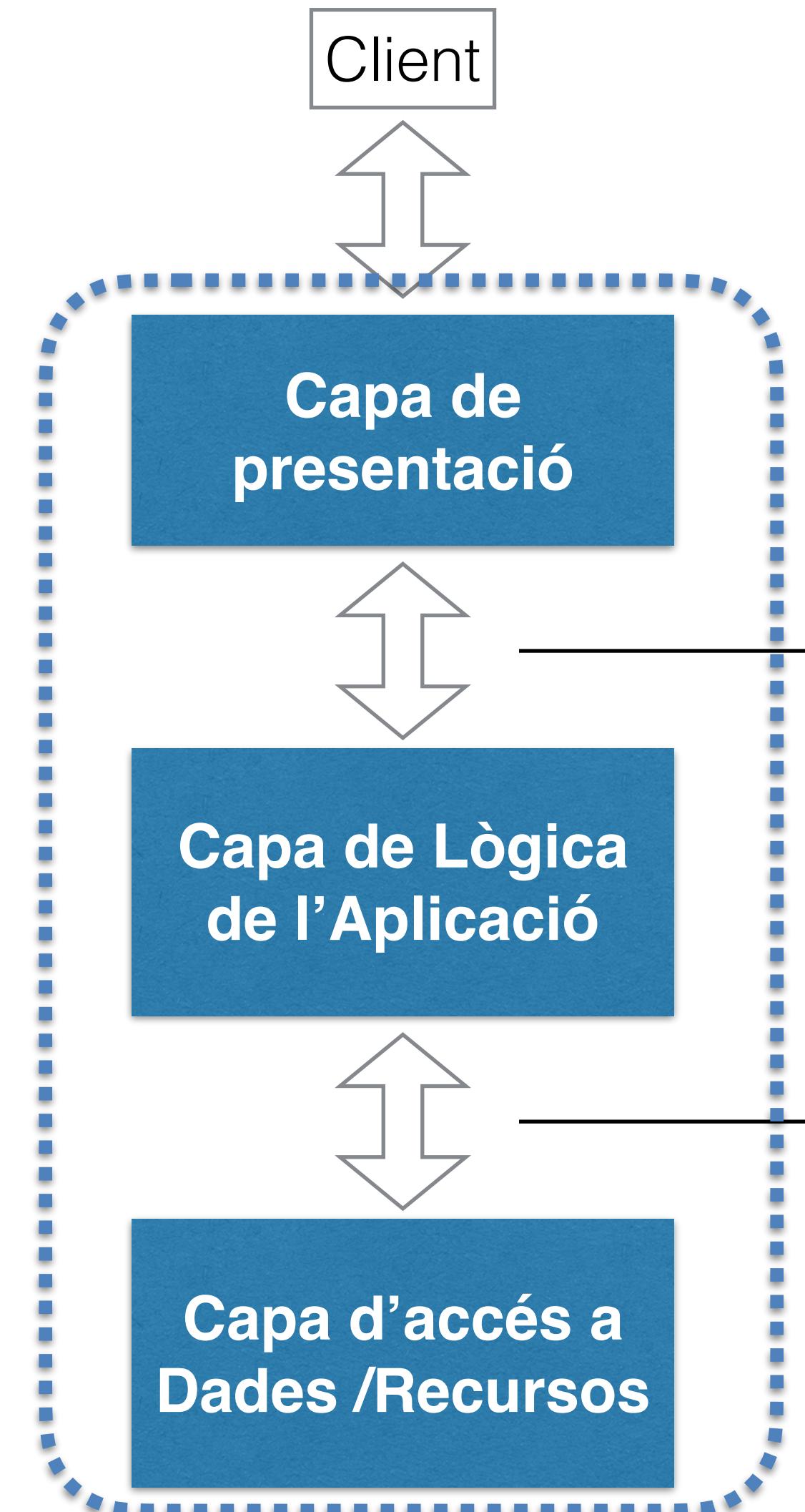
The singleton pattern ensures that only one object of a particular class is ever created. All further references to objects of the singleton class refer to the same underlying instance.

3.1. Introducció

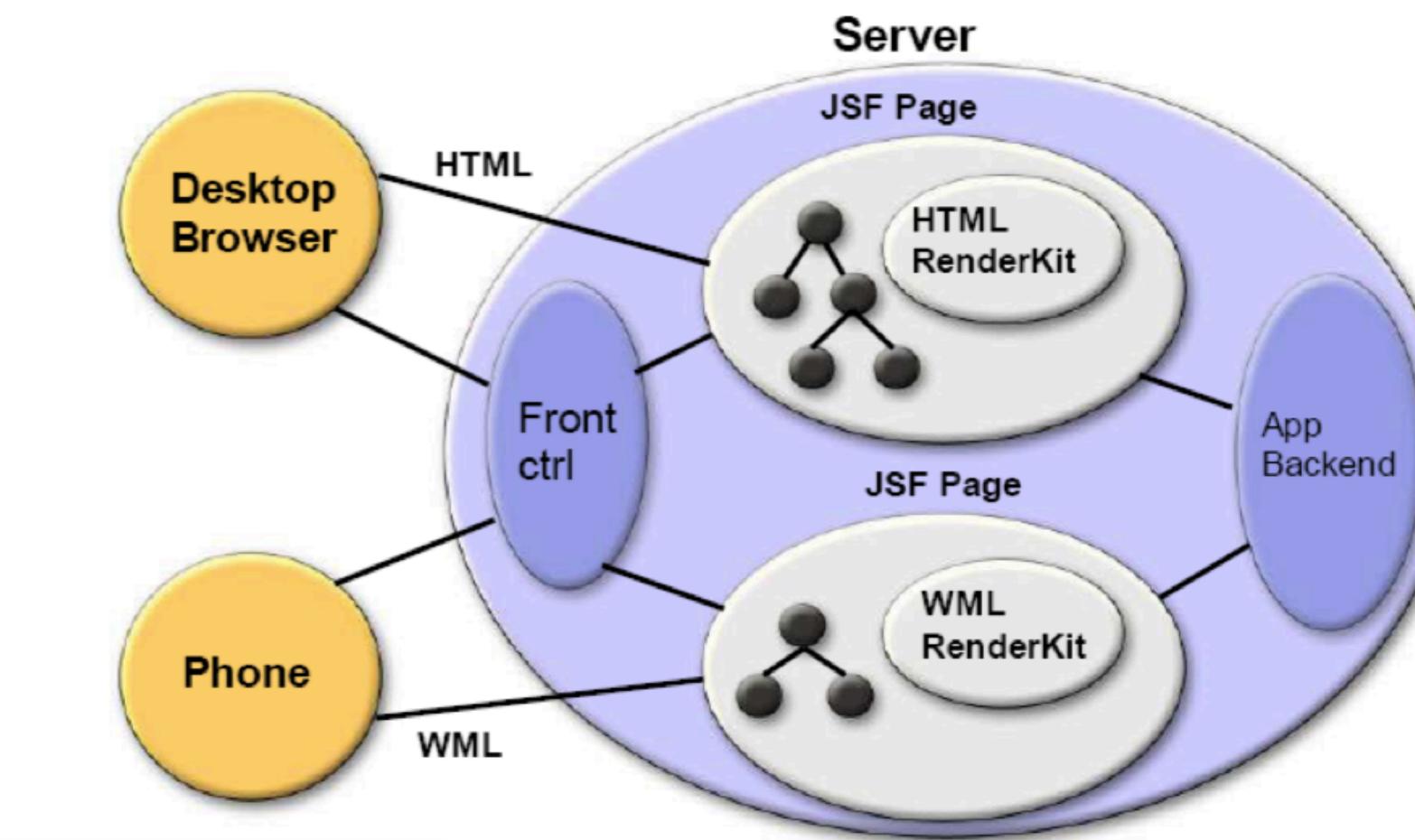


3.2. Patrons arquitectònics

Patró per capes:



Patró Model-Vista-
Controlador



Patró Data Access
Object



HIBERNATE



JDBC Templates

3.1. Introducció

VISTA:

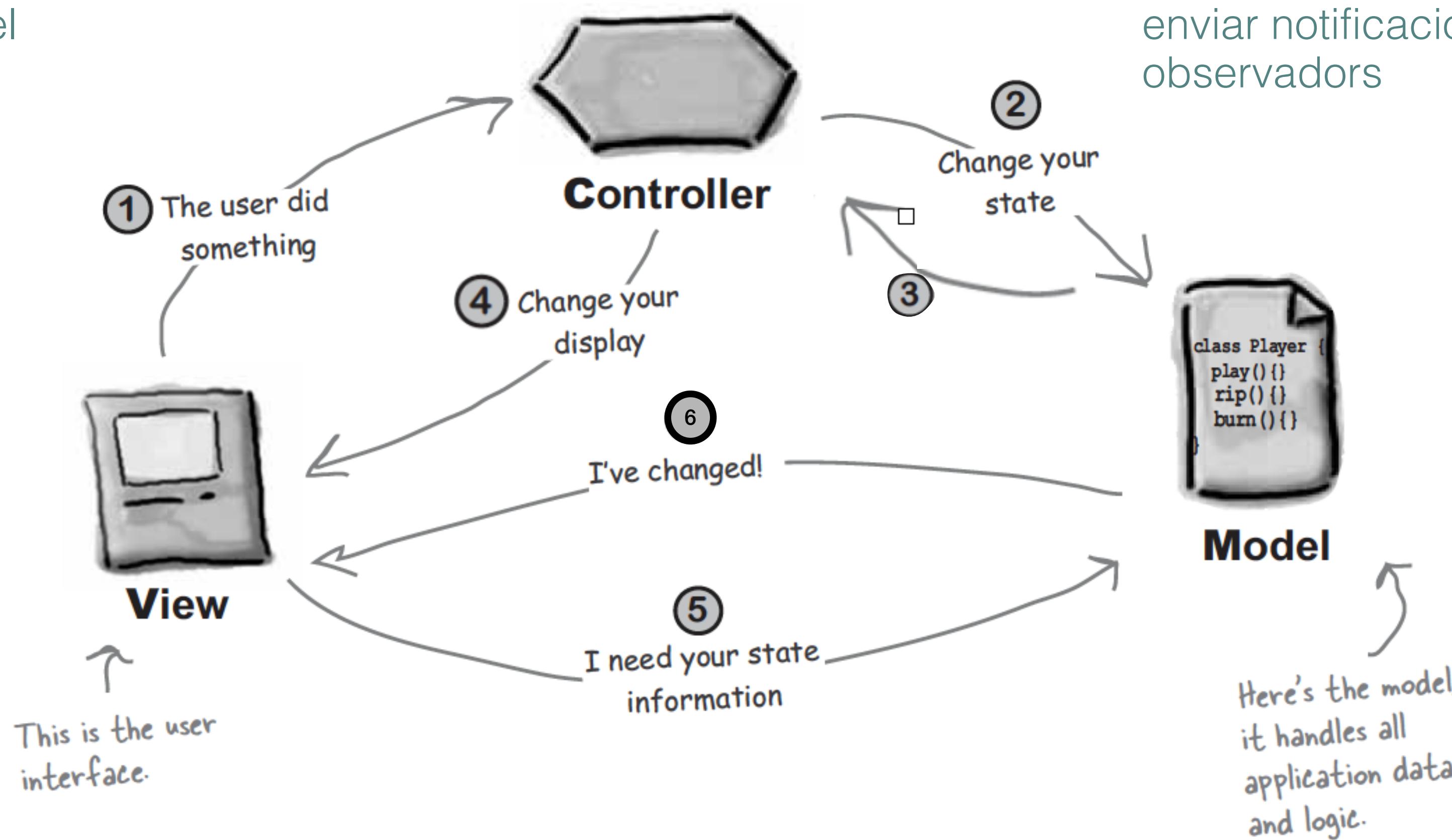
Dóna la presentació del model. La vista normalment mostra l'estat de les dades i el seu valor directament del model

CONTROLADOR:

Agafa l'entrada de l'usuari i li dóna el què significa al model

MODEL:

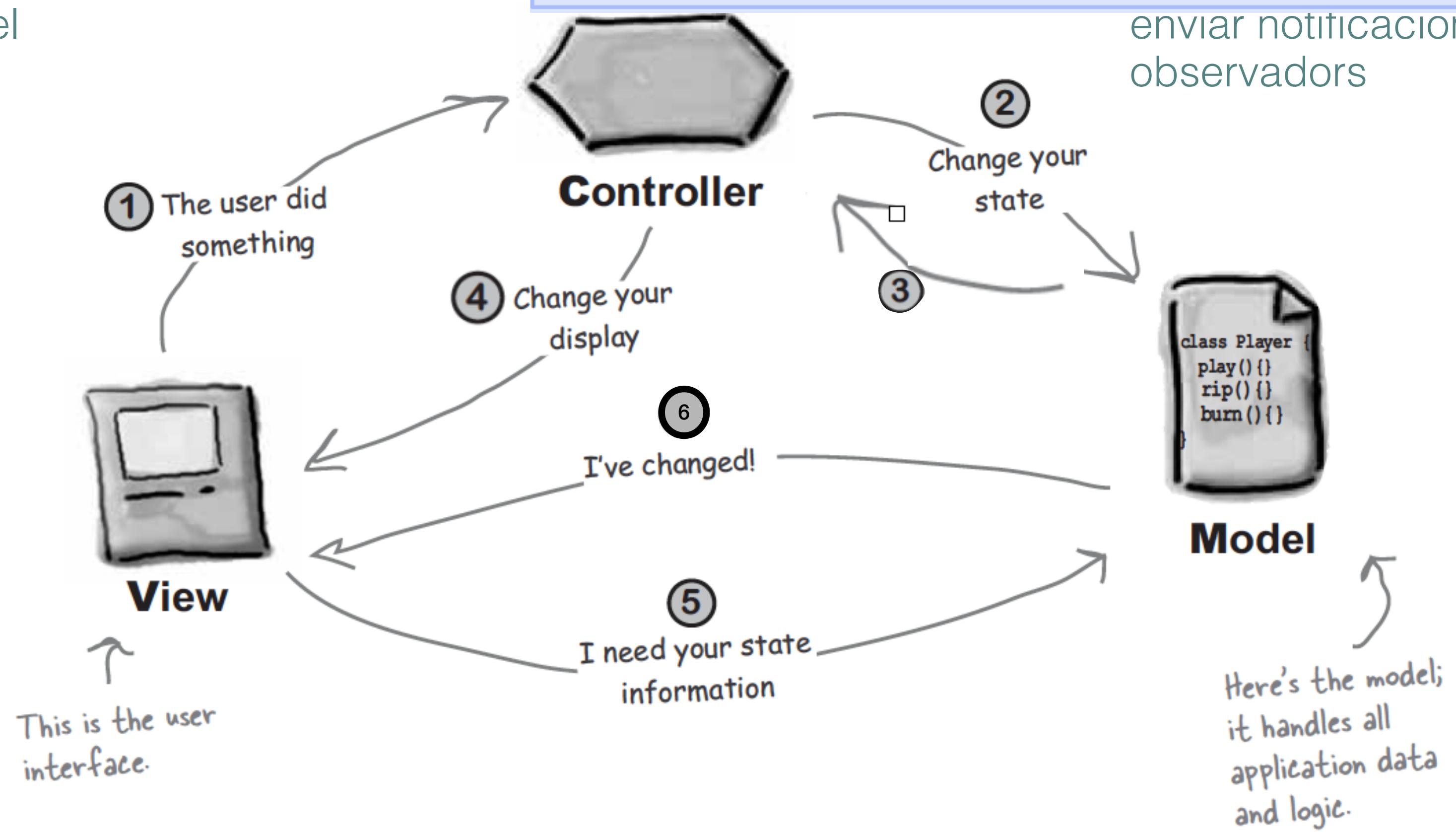
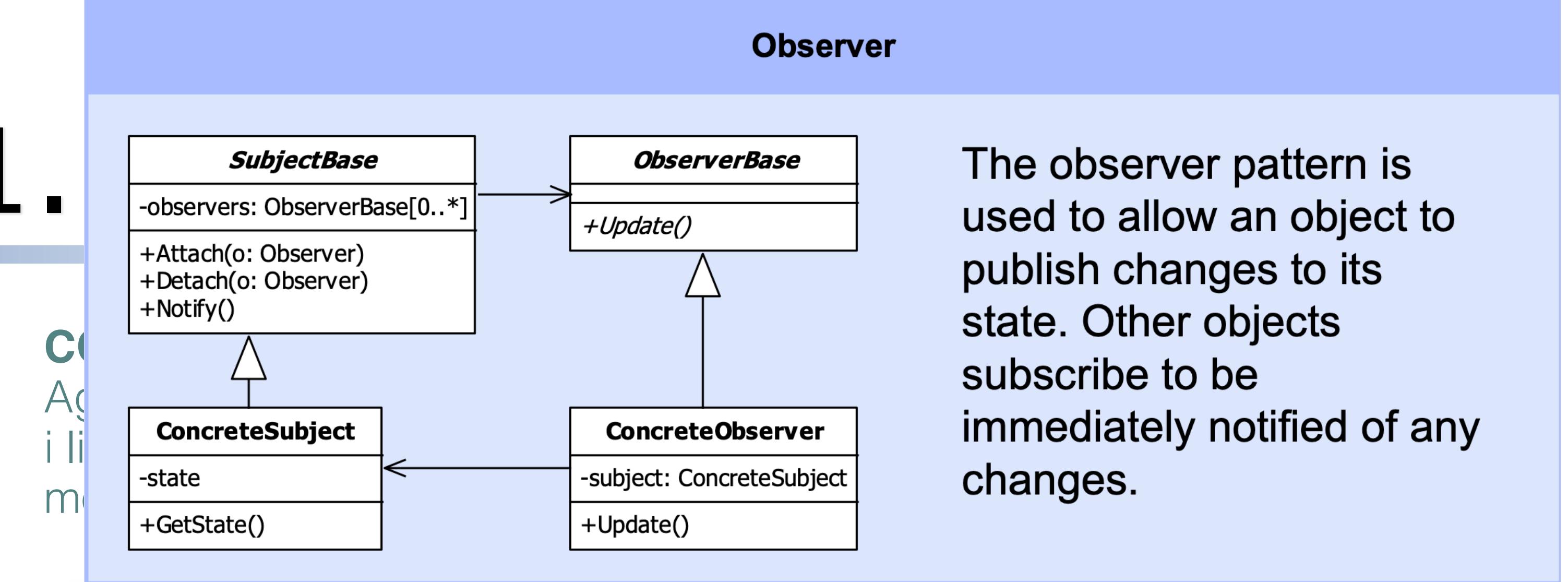
El model guarda totes les dates, l'estat i la lògica de l'aplicació. Dóna una interfície per manipular i donar el seu estat i pot enviar notificacions als observadors



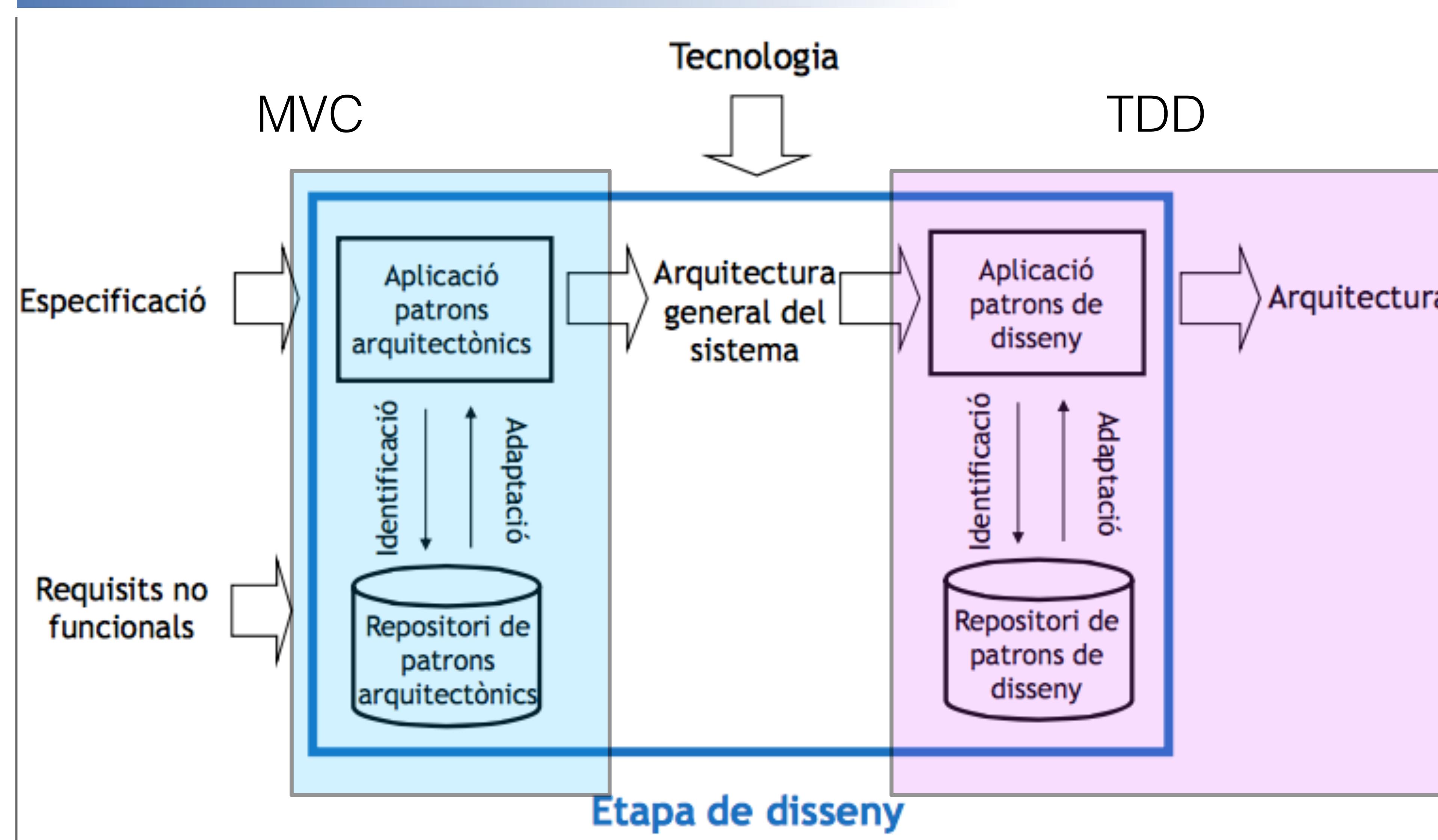
3.1.

VISTA:

Dóna la presentació del model. La vista normalment mostra l'estat de les dades i el seu valor directament del model



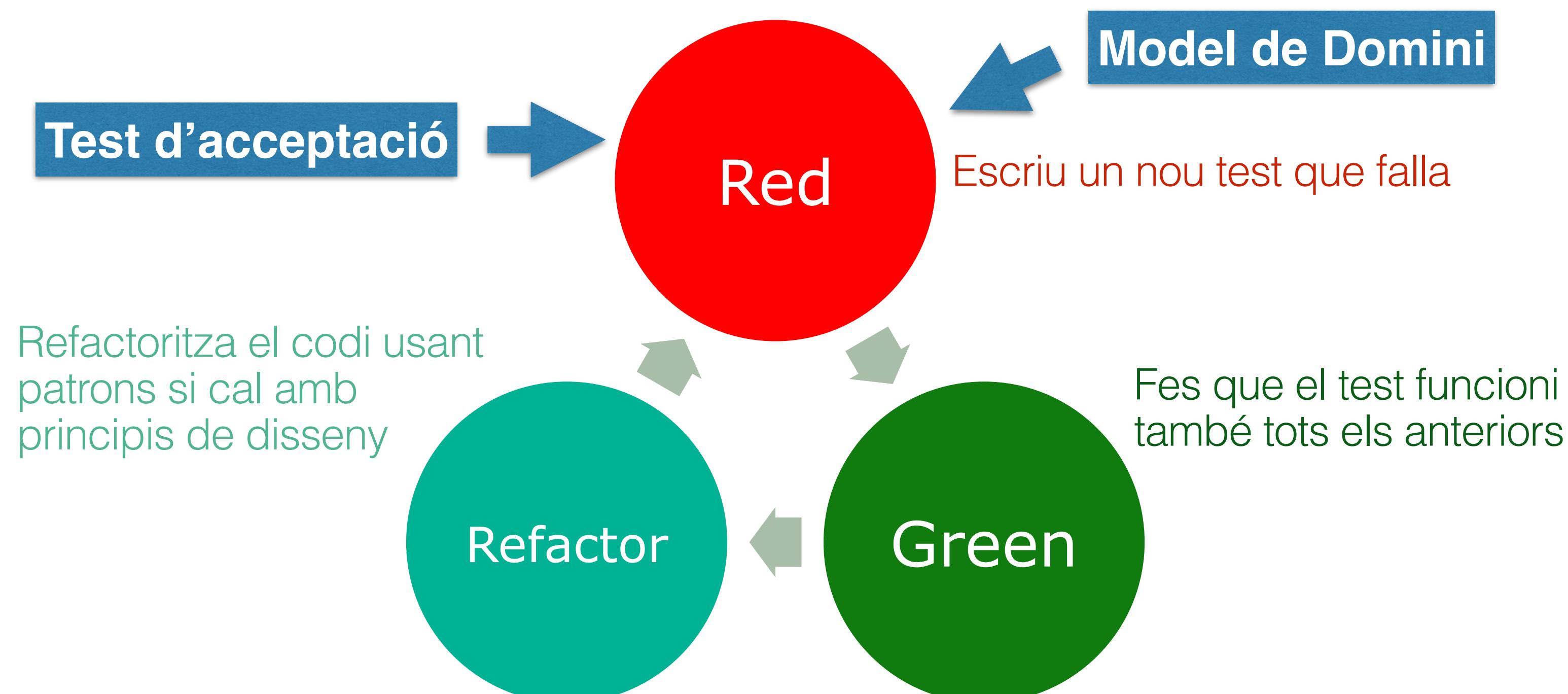
3.1. Introducció



3.1. Introducció

TDD (Test Driven Development): Basat en **dues** senzilles regles:

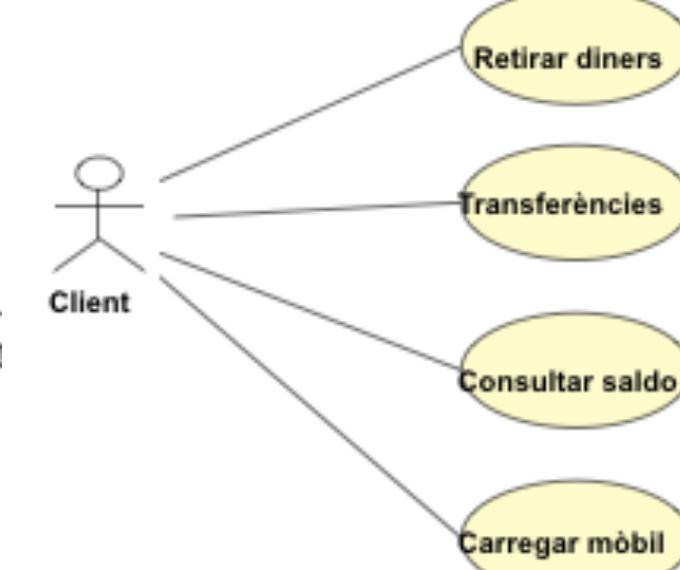
1. Escriu el nou codi només si el test automàtic ha fallat (**Disseny i Implementació**)
2. Elimina la duplicació de codi. (**Disseny**)



3.1. Introducció

Escenari simple de ProcesarVenta para el pago en efectivo

1. El Cliente llega al terminal PDV.
2. El Cajero inicia una nueva venta.
3. El Cajero inserta el identificador del artículo.
4. El Sistema registra la línea de venta y presenta la descripción del artículo, precio y suma parcial.
- El Cajero repite los pasos 3 y 4 hasta que se indique.
5. El Sistema muestra el total con los impuestos calculados.
6. El Cajero le dice al Cliente el total, y pide que le pague.
7. El Cliente paga y el Sistema gestiona el pago.
- ...



Com a [rol d'usuari]
vull [objectiu]
per què així [raó]

En cas que [context]
quan [event]
el sistema [resultat]

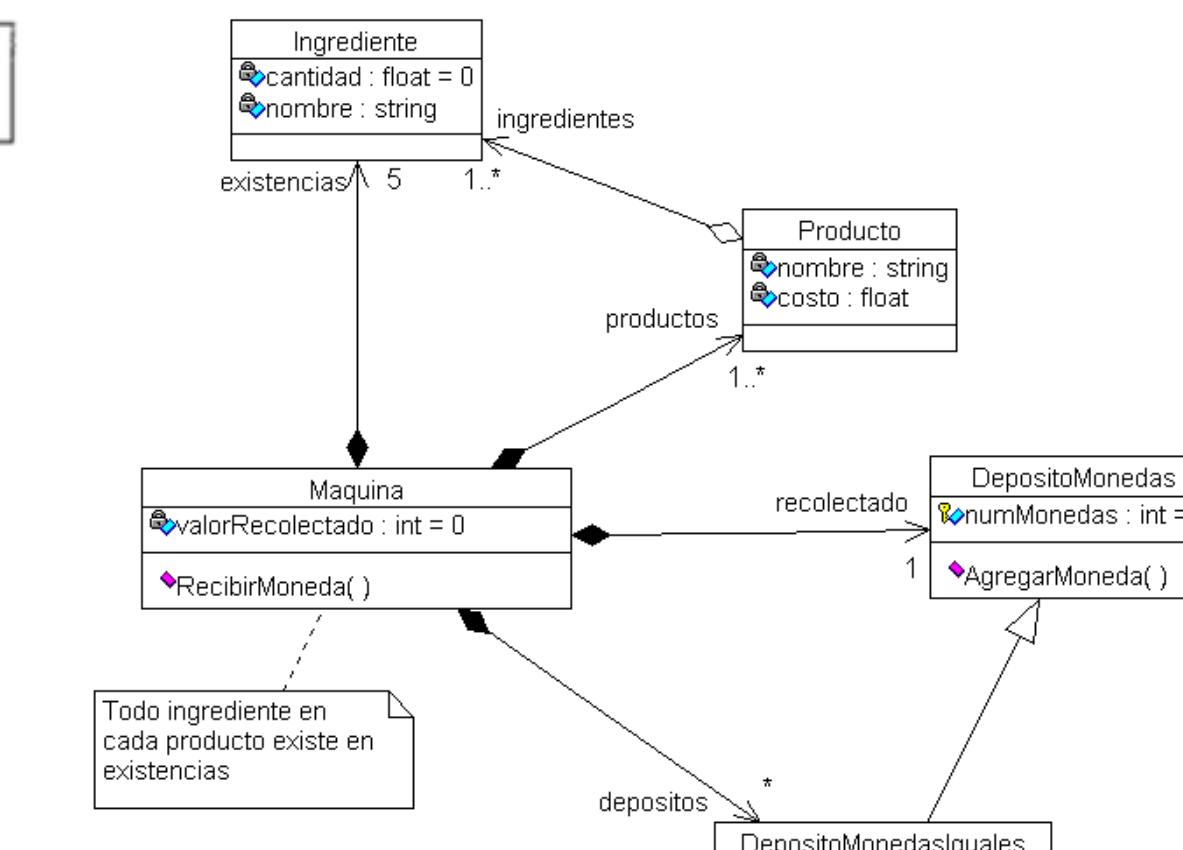
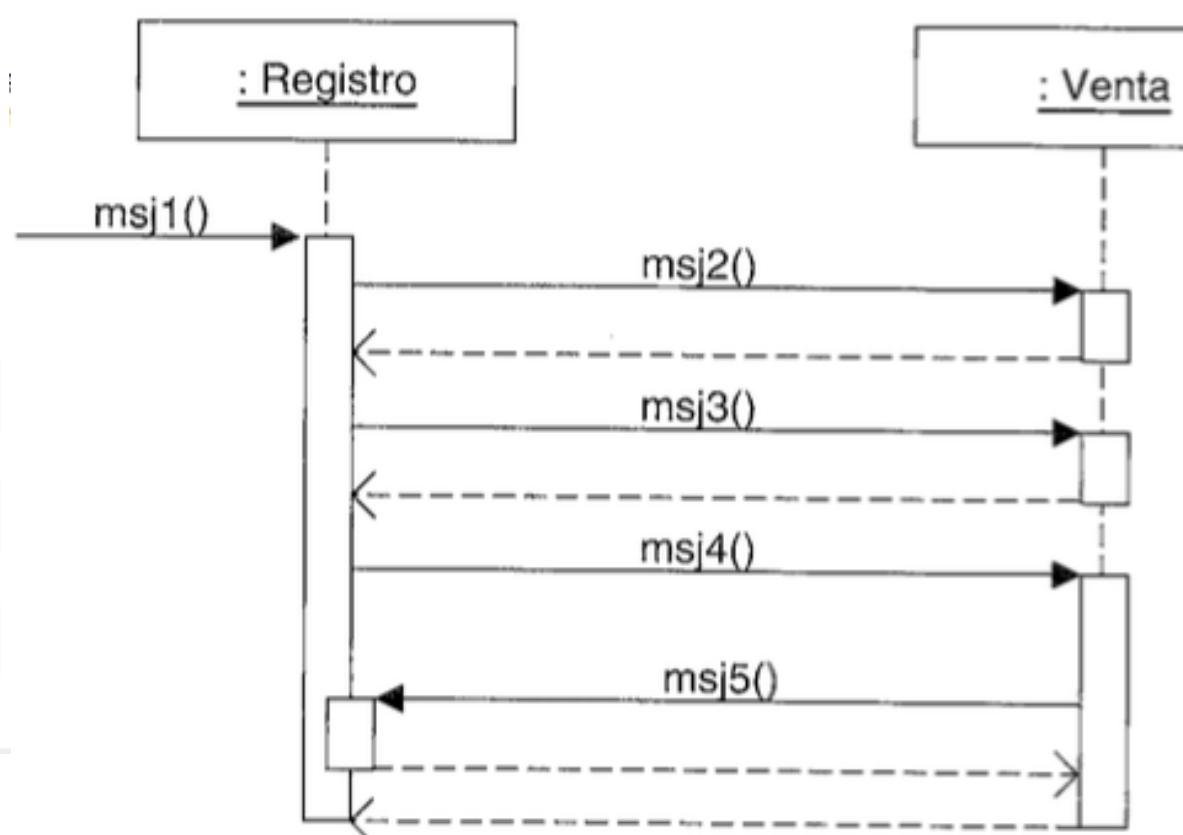
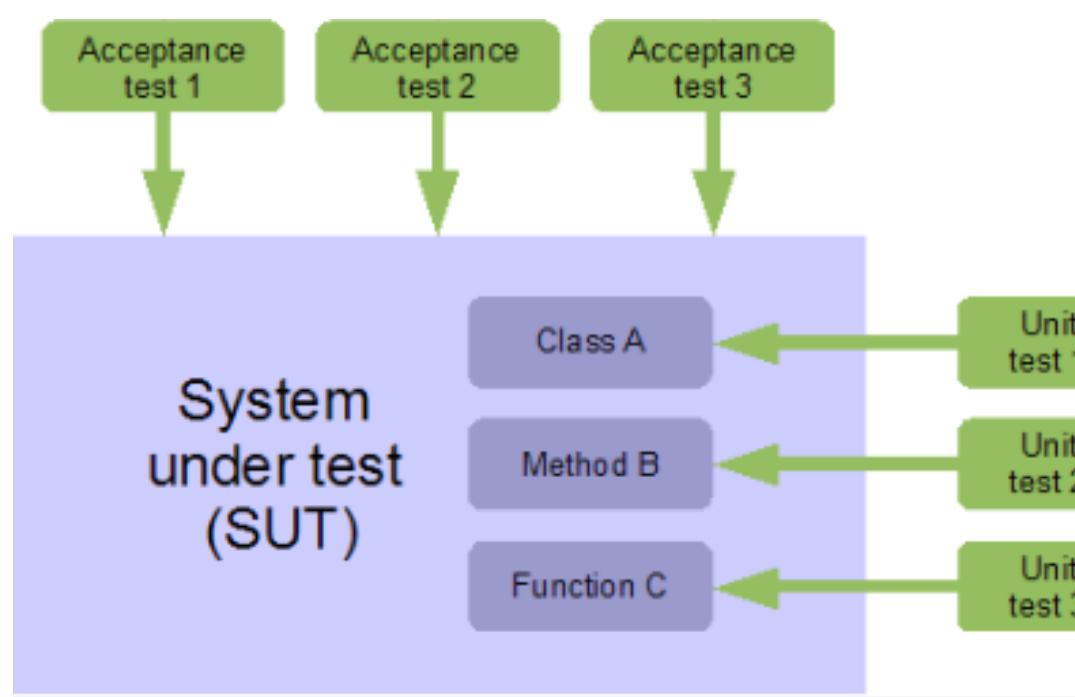
Casos d'ús

DCU

**User Stories+Criteris
d'acceptació**



Model de Domini



**Tests d'acceptació +
Tests unitaris**

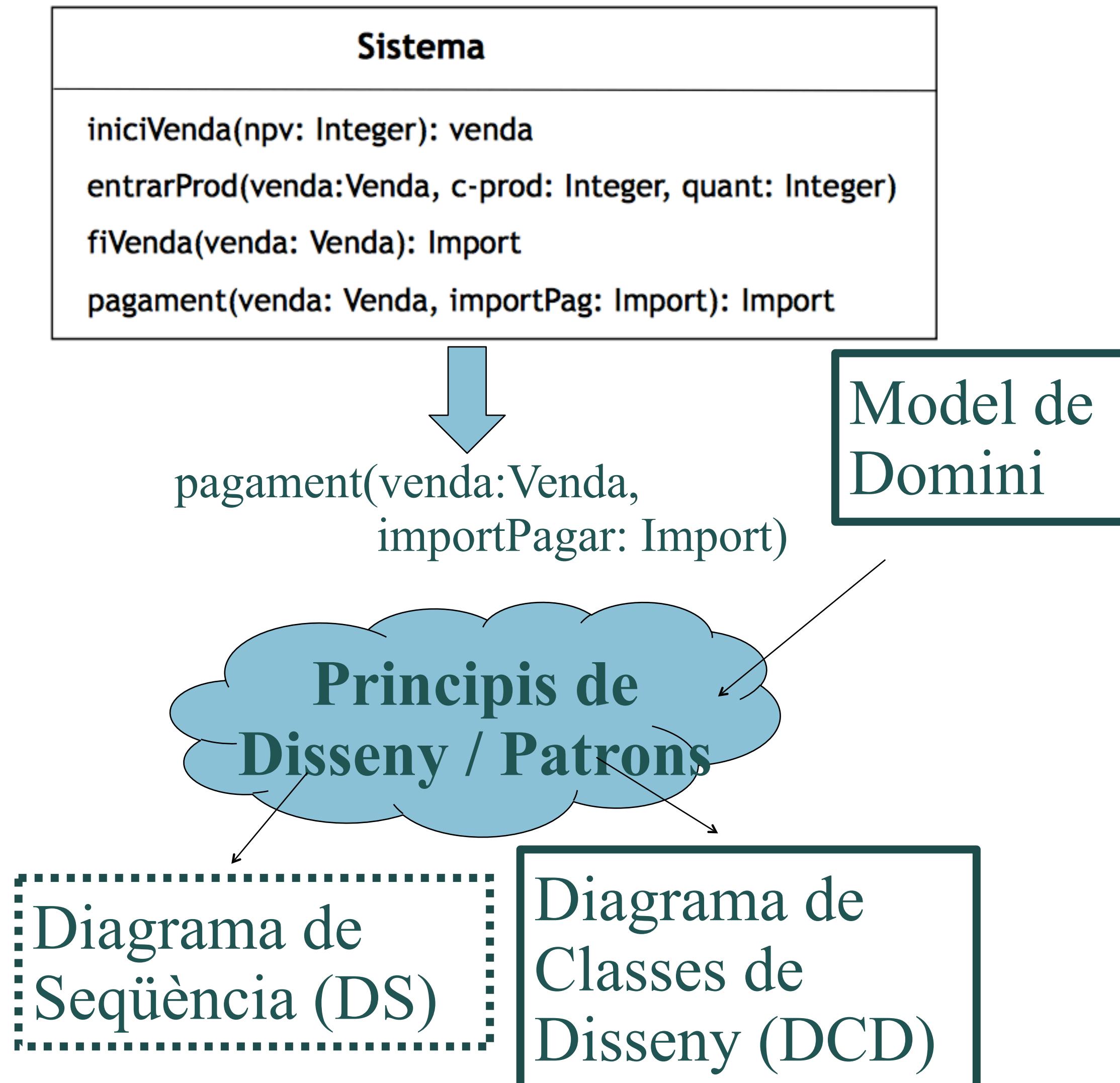
Diagrama de Seqüència (DS)

**Diagrama de Classes de
Disseny (DCD)**

Passos en el disseny

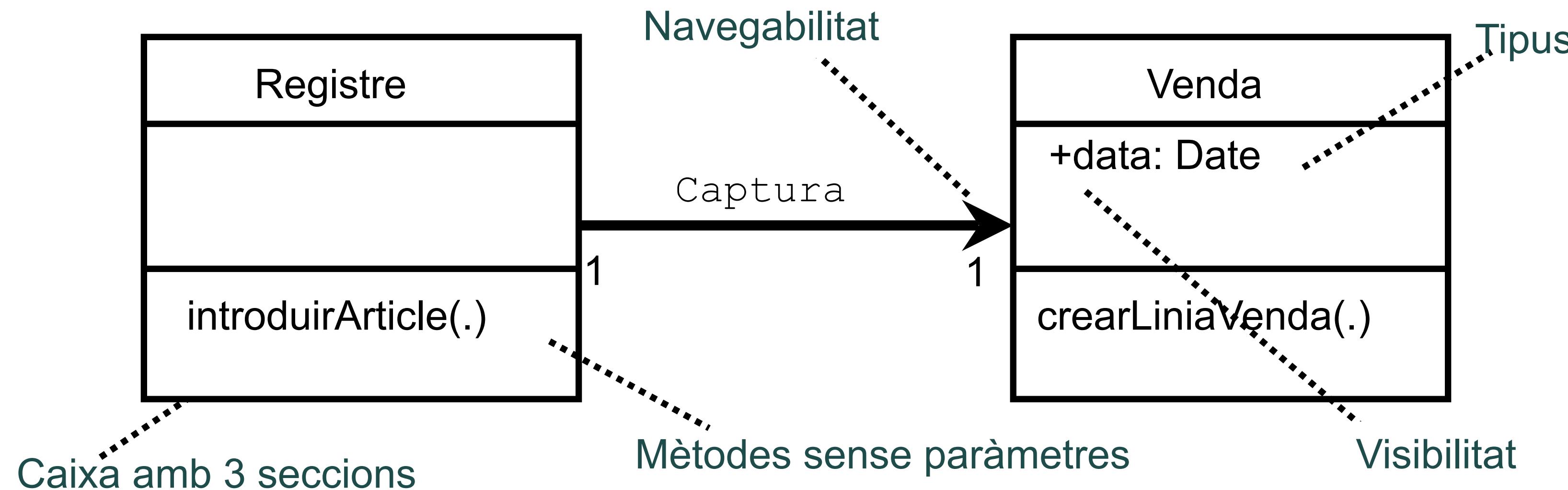
Per a cada **test d'acceptació** definit a l'especificació

1. Es dissenya/pensa el Diagrama de Seqüència (**DS**) del test.
2. A mesura que es necessiten classes en el DS s'afegeixen en el **Diagrama de Classes** a partir, si és possible de les classes conceptuais del **Model de Domini**



3.1. Introducció

Un **Diagrama de Classes de Disseny** (DCD) il·lustra les especificacions per classes software i interfícies en una aplicació



3.1 Exemple senzill

- El joc de daus

Jugar a daus: Un jugador llança 2 daus de sis cares. Si el valor de la suma dels punts representats a la cara superior d'ambdós és 7, el jugador guanya; altrament, perd”



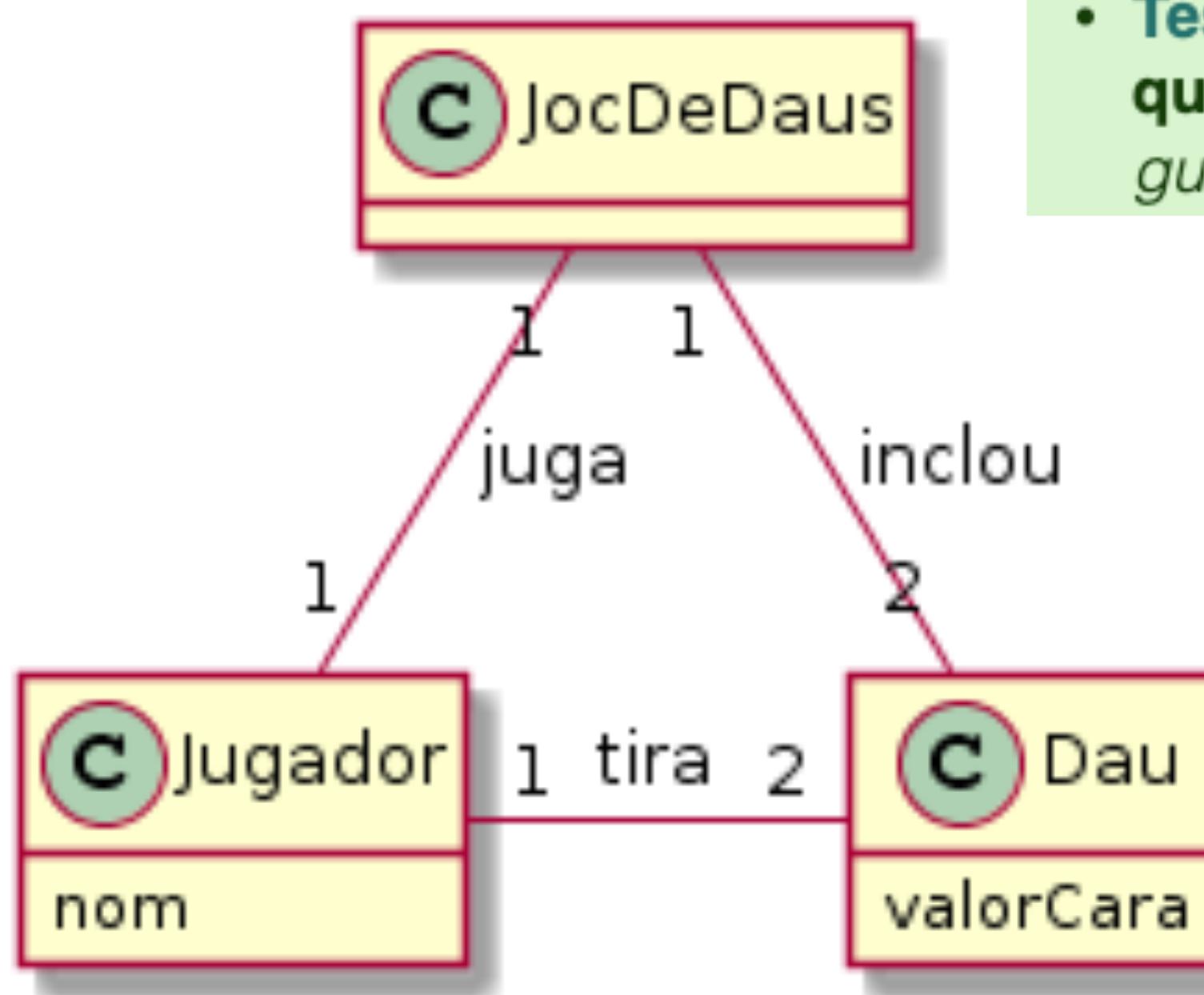
User stories

- **Com a usuari vull** enregistrar-me **per a** poder jugar
- **Com a jugador vull** tirar dos daus **per a** poder saber si he guanyat
 - **Test acceptació 1: En el cas que** els dos daus sumin 7 **quan** el jugador mira la jugada **el sistema** contesta “Has guanyat”
 - **Test acceptació 2: En el cas que** els dos daus no sumin 7 **quan** el jugador mira la jugada **el sistema** contesta “Has perdit”
- **Com a jugador vull** saber el millor resultat **per a** poder comparar-me amb d'altres jugadors

Model de domini

- Com a jugador vull tirar dos daus per a poder saber si he guanyat

- Test acceptació 1: En el cas que els dos daus sumin 7 quan el jugador mira la jugada el sistema contesta “Has guanyat”



Diagrams d'interacció

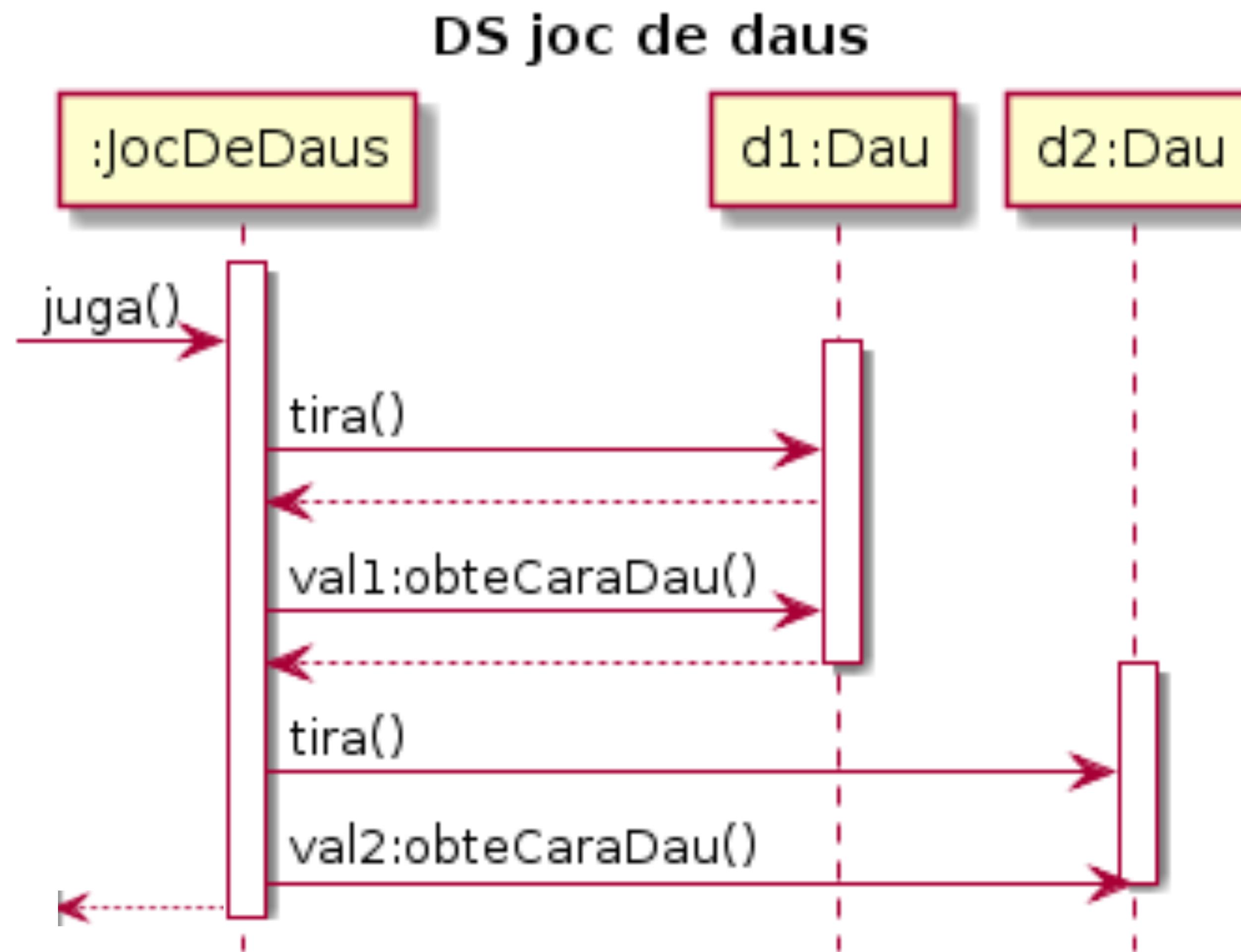
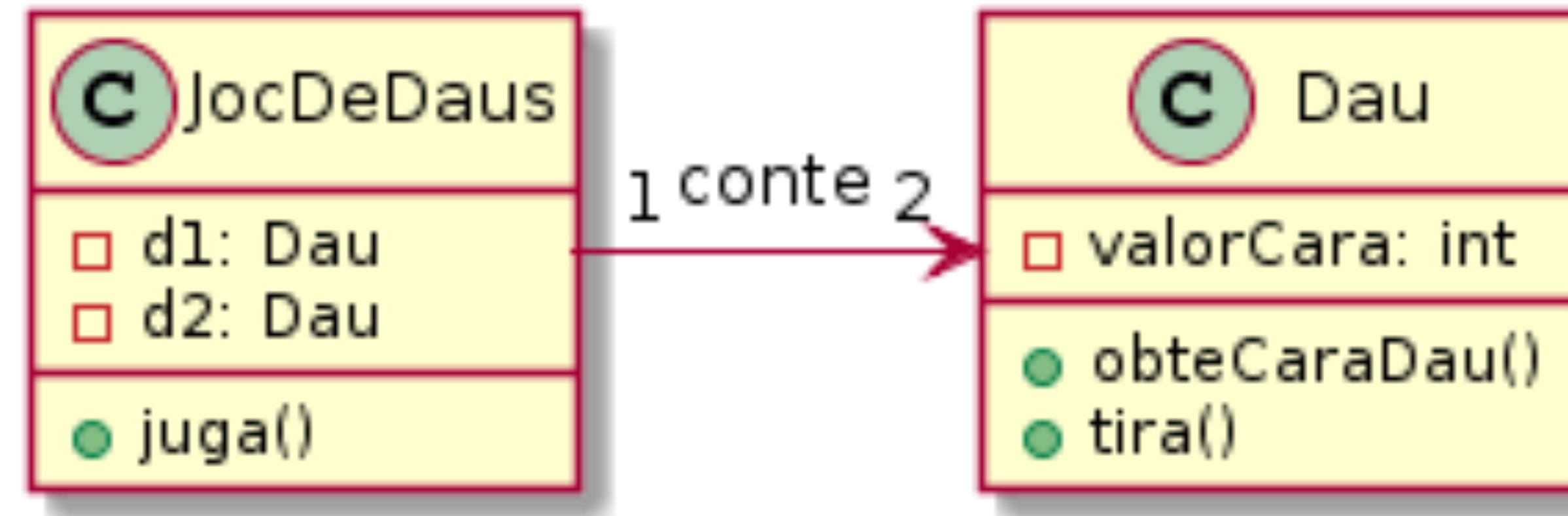


Diagrama de Classes de Disseny



Passos per crear el Diagrama de Classes

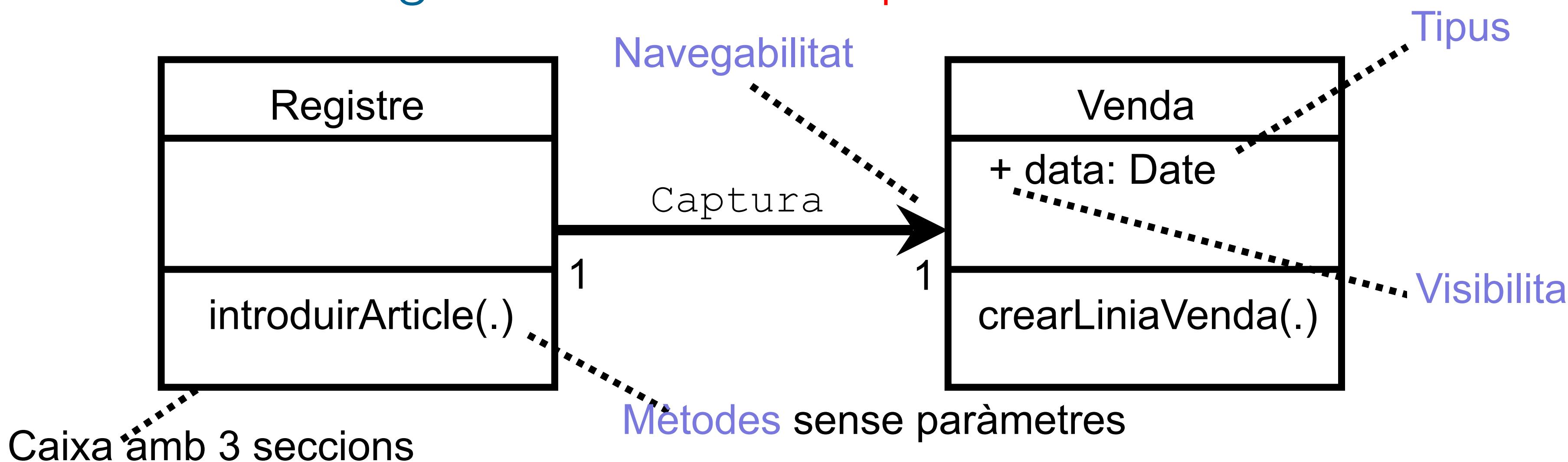
Pas 1. Identificar classes i il·lustrar-les

Pas 2. Afegir els noms dels mètodes

Pas 3. Afegir informació de tipus i visibilitat (inclou atributs i paràmetres)

Pas 4. Afegir associacions i navegabilitat

Pas 5. Afegir relacions de dependència



3.1 Exemple senzill

- El joc de daus (versió II)

Jugar a daus: Joc de dos jugadors. Tots dos jugadors tiren un dau i guanya el que treu puntuació màxima



Nova User Story

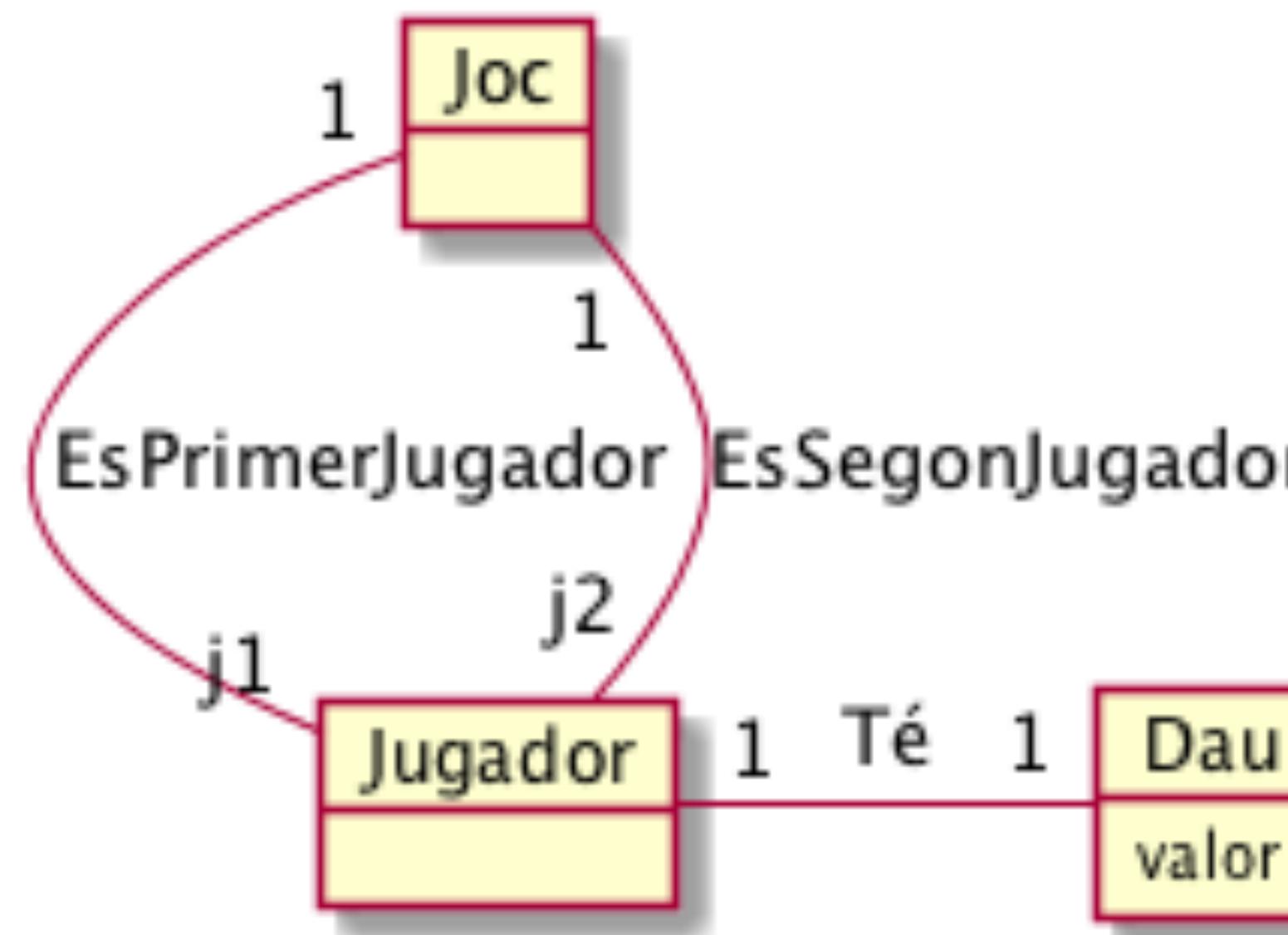
- **Com a** jugador en joc per parelles **vull** jugar amb un altre jugador-me **per a** veure qui treu el dau més alt

- **Test acceptació 1:** **En el cas que** el jugador 1 treu una puntuació més gran **quan** els jugadors miren la jugada **el sistema** contesta “*Ha guanyat el jugador 1*”
- **Test acceptació 2:** **En el cas que** el jugador 2 treu una puntuació més gran **quan** els jugadors miren la jugada **el sistema** contesta “*Ha guanyat el jugador 2*”

Model de domini

- Com a jugador en joc per parelles **vull** jugar amb un altre jugador-me **per a** veure qui treu el dau més alt

MD de Joc de Daus



- **Test acceptació 1:** En el cas que el jugador 1 treu una puntuació més gran **quan** els jugadors miren la jugada **el sistema** contesta “*Ha guanyat el jugador 1*”

Pas 1. Identificar classes

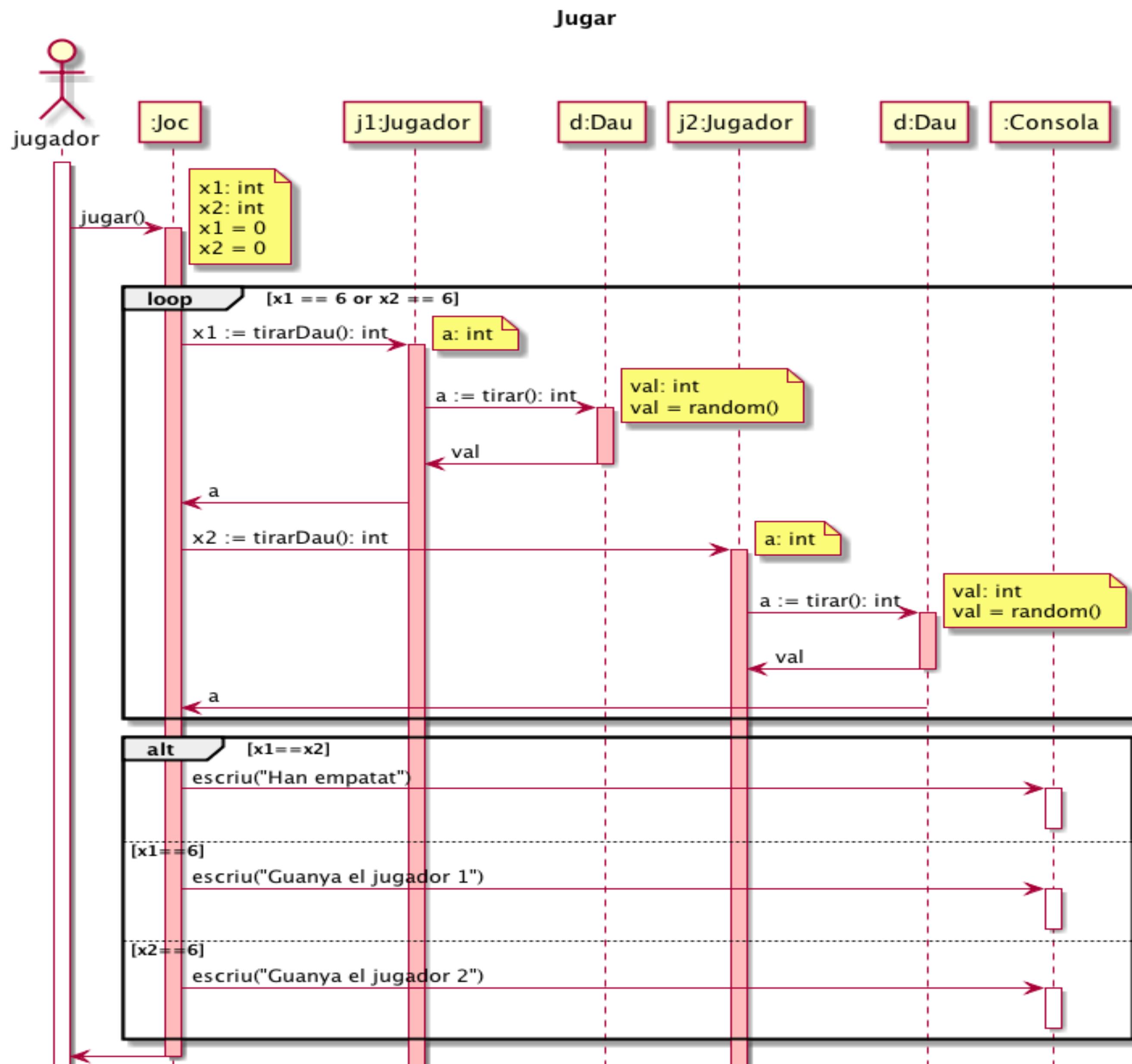
Passos:

1. Trobar **classes** fent un recorregut del test per les classes i llistant les classe que semblen que puguin estar implicades
2. Dibuixar un diagrama de classes per les classes trobades, incloent-hi els **atributs** identificats per aquestes classes en el model de domini



No totes les classes de domini es convertiran en classes de disseny

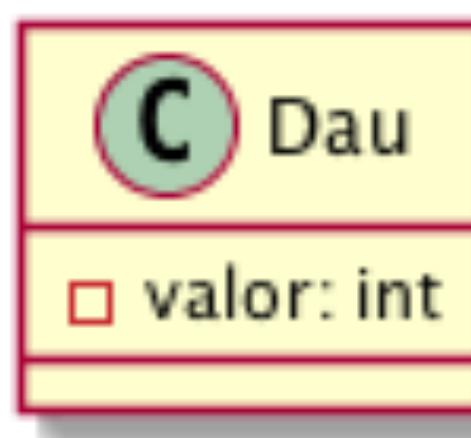
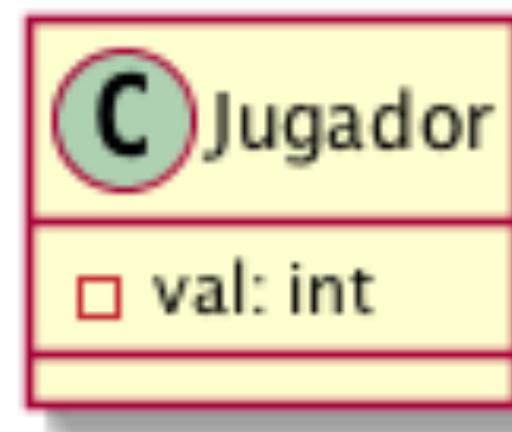
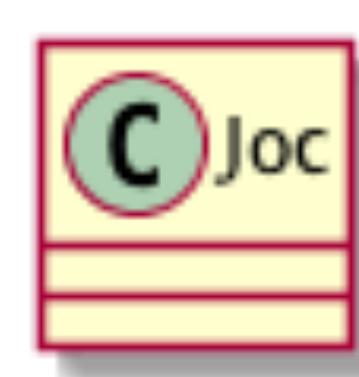
Exemple Joc de Daus



Exemple

Classes i atributs identificades en el Joc de Daus

DCD-Pas 1 de Joc de Daus



Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◊	◊	protected
~	△	△	package private
+	○	●	public

Pas 2. Afegir els noms dels mètodes

- El missatge **create**. Els constructors i destructors normalment no apareixen en el diagrama
- Mètodes d'accés a atributs (**setter/getter**). Habitualment s'omet
- Missatges a multi-objectes (collections). No s'han d'afegir

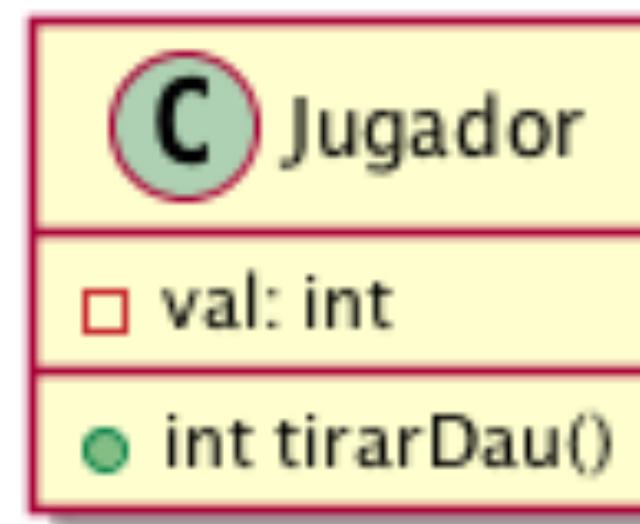
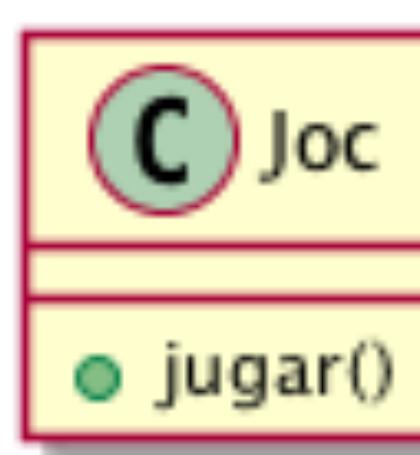
Pas 3. Afegir informació de tipus i visibilitat

- Es pot afegir informació dels tipus dels atributs, paràmetres i valors de retorn dels mètodes
- Si el diagrama es crea per a una eina CASE (com EclipseUML, Rational Rose, ArgoUML o Sketch it! de IntelliJ) amb generació automàtica de codi, s'han de repassar de forma exhaustiva tots els detalls
- Si el diagrama es crea per a que altres desenvolupadors puguin llegir-lo, un excessiu nivell de detall pot impactar negativament la facilitat de comprensió del diagrama (tot i que és preferible passar-se a quedar-se curt)

Exemple

Mètodes identificats en el Joc de Daus i visibilitat associada¹

DCD de Joc de Daus Passos 2 i 3



```
@startuml  
title DCD de Joc de Daus Passos 2 i  
3  
...  
  
Joc : +jugar()  
Jugador : +tirarDau()  
Dau : +tirar()  
...  
@enduml
```

Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◊	◊	protected
~	△	△	package private
+	○	●	public

Pas 4. Afegir associacions i navegabilitat

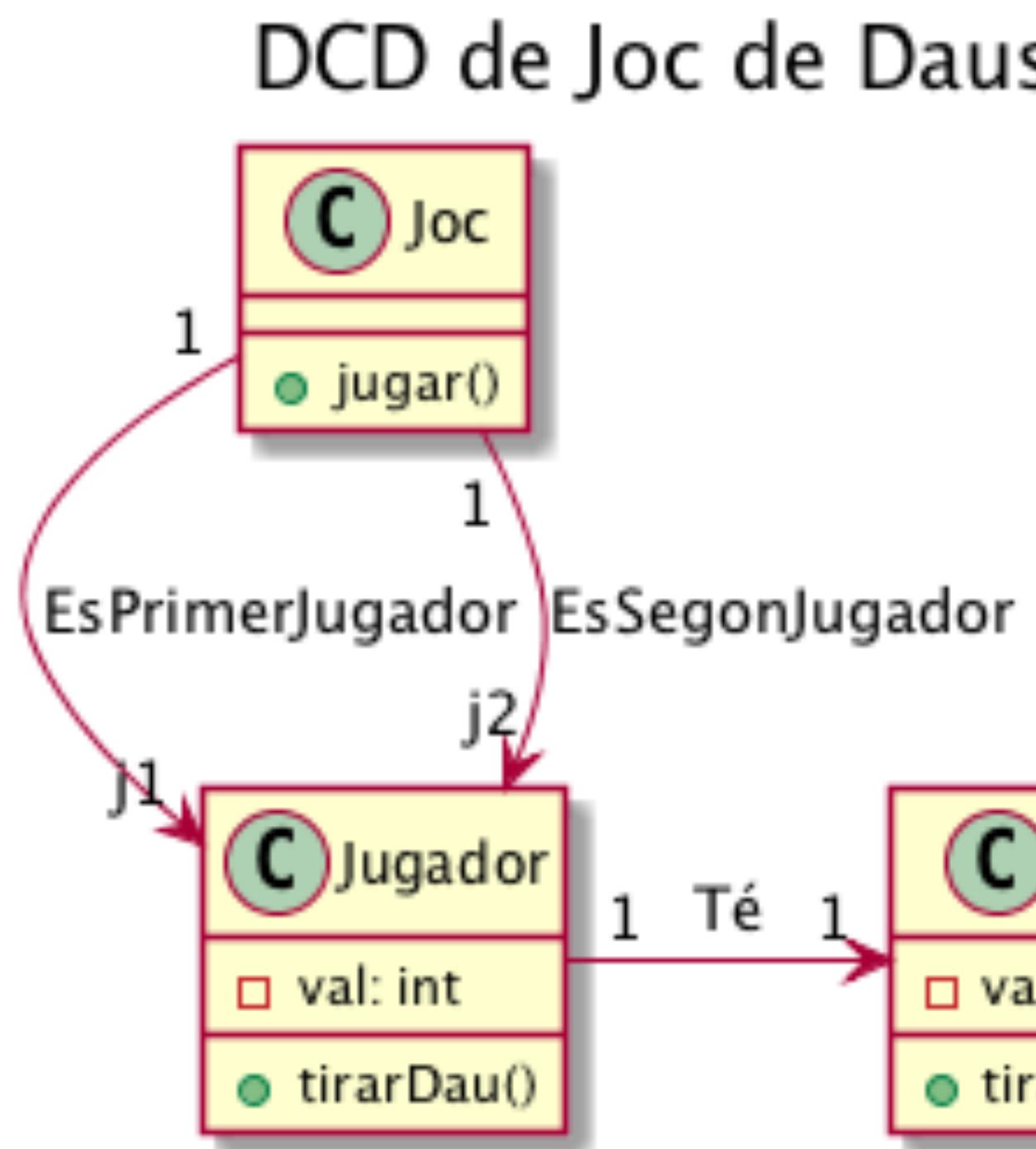
- **Navegabilitat:** és una propietat d'un rol d'una associació que ens indica que és possible navegar unidireccionalment d'objectes de l'origen a objectes del destí segons la direcció indicada per la fletxa
- En el DCD cada rol es pot decorar amb una fletxa de navegabilitat. Per a la majoria de les associacions és molt important indicar la navegabilitat
- La navegabilitat indica també visibilitat. Habitualment, visibilitat per atribut i es posa el nom del rol del model de domini

Pas 5. Afegir relacions de dependència

- UML inclou una relació de dependència genèrica que indica que un element de qualsevol tipus (classes, casos d'ús, etc.) té coneixement d'un altre
- En els DCD pot ser molt útil representar visibilitat entre classes que no siguin per atribut (si és per atribut s'usa una associació), ja sigui paràmetre, global o local.

Exemple

Pas 4: Associacions i Navegabilitat en el Joc de Daus



```
@startuml
title DCD de Joc de Daus Pas 4
...
Joc "1"-down->"j1" Jugador: EsPrimerJugador
Joc "1"-down->"j2" Jugador: EsSegonJugador
Jugador "1"-right->"1" Dau: Té
...
@enduml
```

Activitat: Obriu el IntelliJ del projecte Daus i mireu el plantUML que genera el Scketch it!!
Compara'l amb el DCD retocat que tens dins de la carpeta src

3.1. Introducció

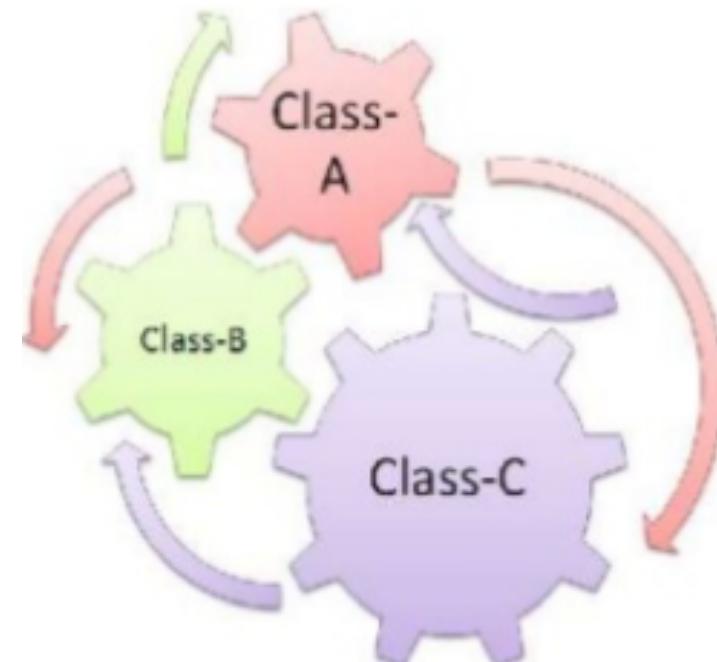
Aspectes que denoten un disseny “dolent” (*code smell*)

- **Rigidesa**: l'impacte d'un canvi en el software és impredecible (cada canvi produceix una cascada de canvis en moltes altres classes o el codi és tan complicat que costa entendre'l)
- **Inmobilitat**: No es pot reutilitzar codi o parts de codi
- **Fragilitat**: A cada canvi, el software es trenca en llocs on no hi han relacions conceptuais
- **Viscositat**: Impossibilitat de canviar el codi sense canviar el disseny. Provoca que fer un pedaç addicional al codi és més fàcil que no pas canviar tot el disseny



Dependències entre classes

3.3.1.Baix acoblament/Alta cohesió



- **Acoblament:**

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

- És necessari veure'l amb ajuda d'altres criteris o patrons
- Acoblar objectes “globals” estables no és un problema

Acoblament BAIX

Classes amb:

- pocs mètodes
- número petit de línies de codi
- no fan gaire feina
- feina relacionada

- **Cohesió:**

mesura del grau de relació i de concentració de les diverses responsabilitats d'una classe (atributs, associacions, mètodes,...)



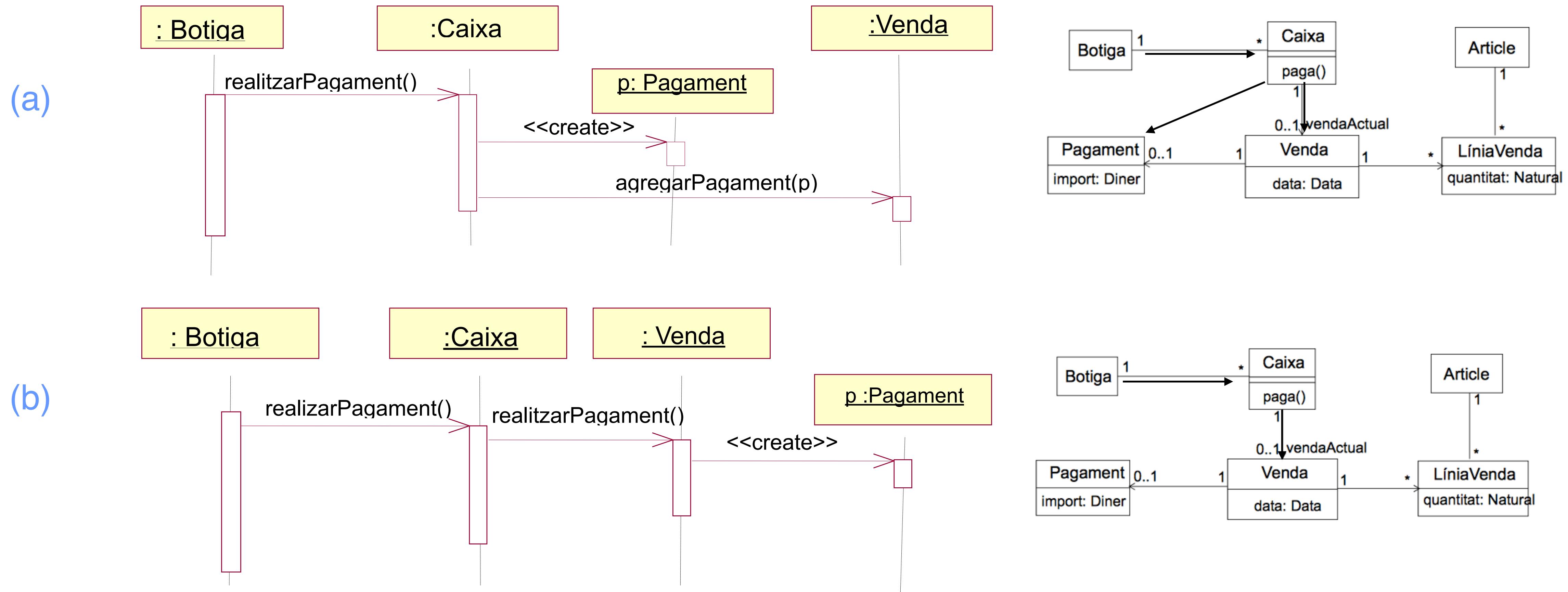
ALTA Cohesió

WARNING

Pot donar lloc a més indireccions (o més crides)

Exemple d'aplicació

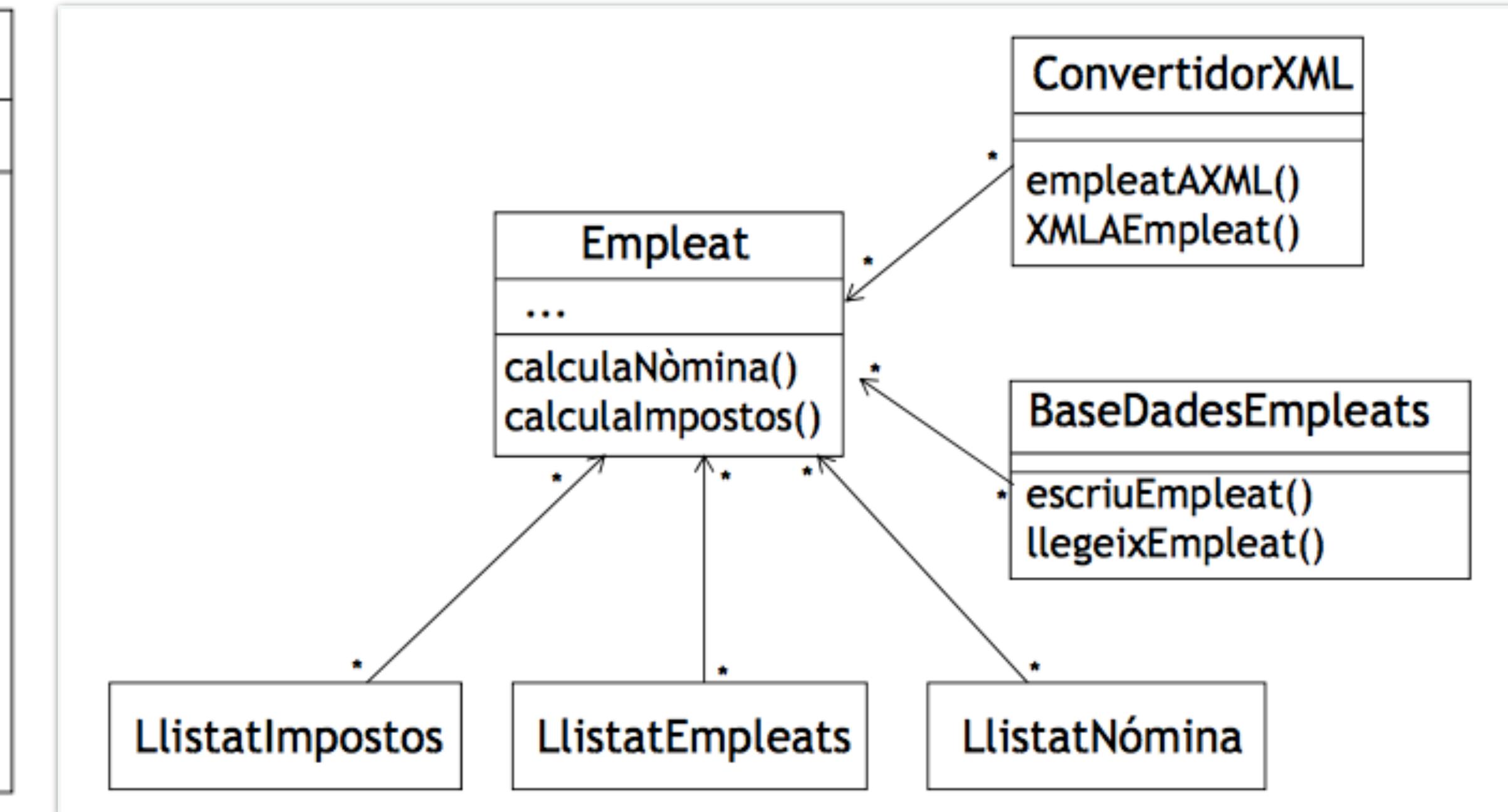
- Què podem dir d'aquests dissenys?



Exemple cohesió

Empleat
...
calculaNòmina()
calculalImpostos()
escriuADisc()
llegeixDeDisc()
creaXML()
llegeixDeXML()
mostraEnLlistatNòmina()
mostraEnLlistatImpostos()
mostraEnLlistatEmpleats()

BAIXA Cohesió



ALTA Cohesió