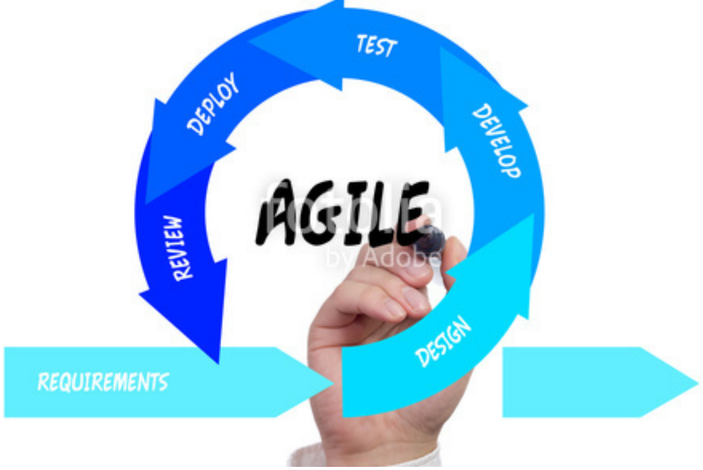


Classe 04.10.2021: Model de Domini

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2021/22

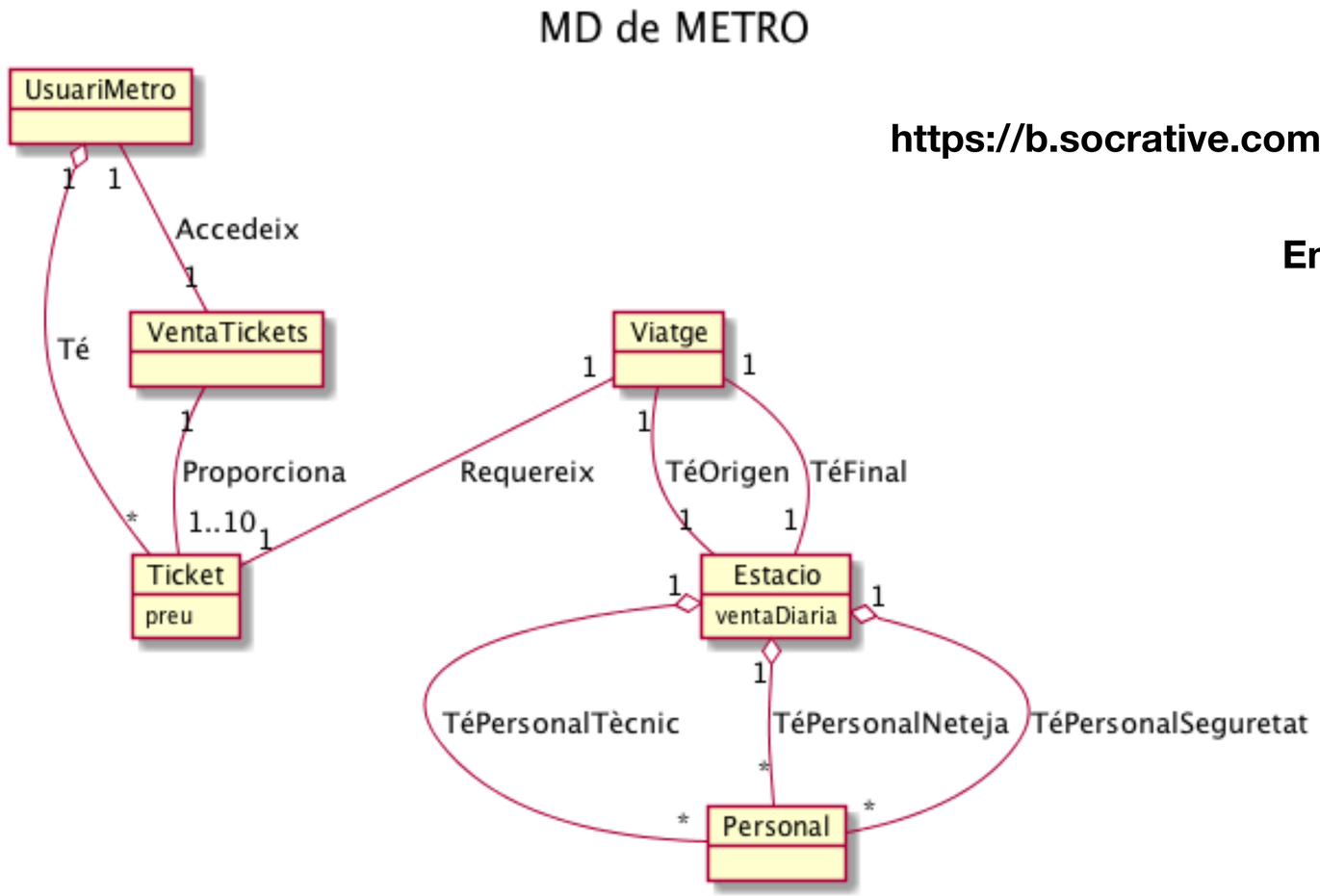
UNIVERSITAT DE BARCELONA

Temari

1. Introducció al procés de desenvolupament del software
2. Anàlisi de requisits i especificació
   - 2.1 Anàlisi de requisits:Model FURPS+
   - 2.2 Especificació: Casos d’ús i User Stories
   - 2.3 Especificació: Model de Domini
3. Disseny
4. Del disseny a la implementació
5. Ús de frameworks de testing

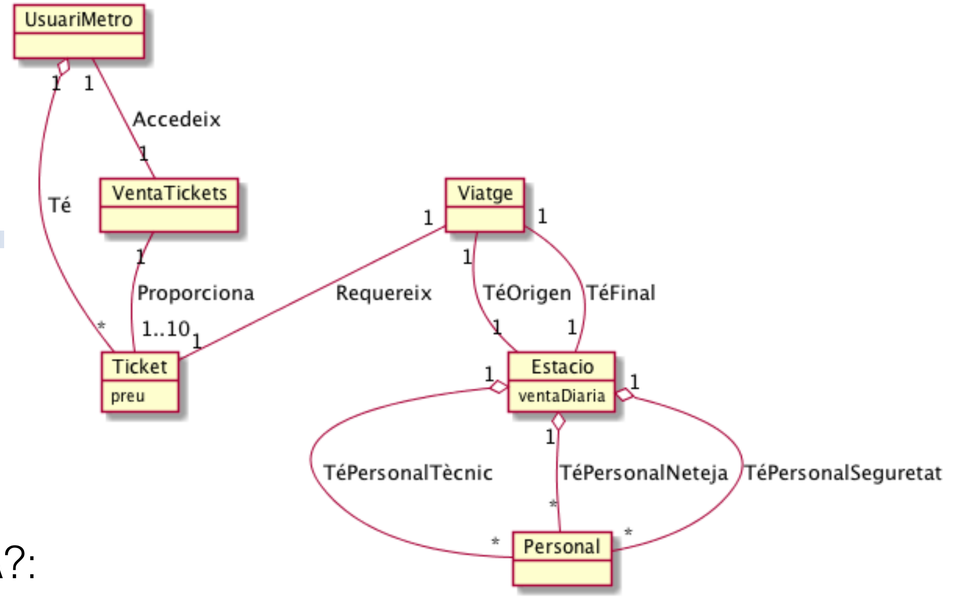
Exercicis Model de Domini

Donat el següent Model de Domini que ha definit un dissenyador de software per a gestionar una estació de metro d'una ciutat qualsevol on es venen tickets individuals fins a un màxim de 10.



https://b.socrative.com/login/student/

Entrar com DS1



Contesta a les següents preguntes:

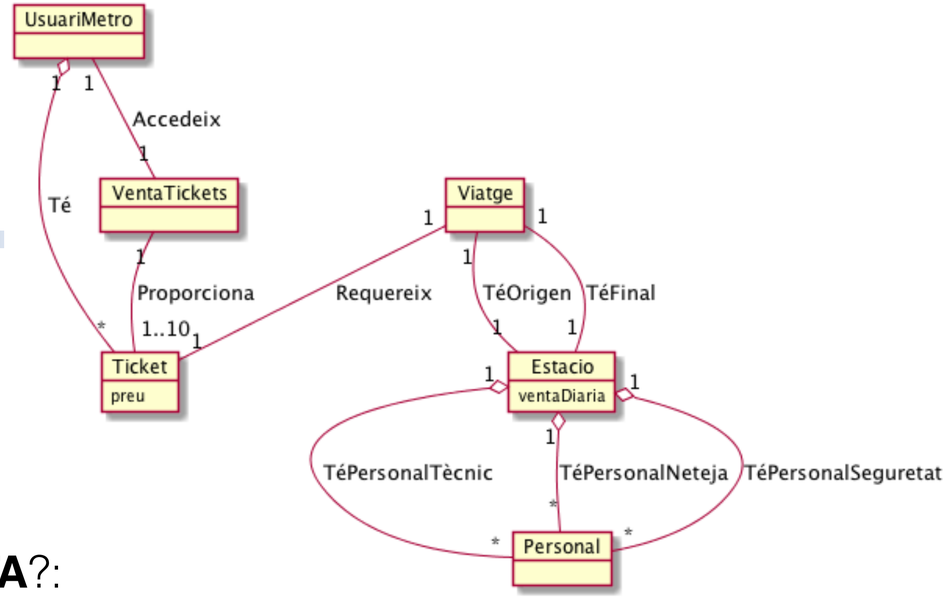
1. Quina de les següents afirmacions és **CERTA**?:

   A. Cal posar Quantitat com tipus de l'atribut "preu" de la classe Ticket per a poder expressar que el preu es pot donar en diferents divises.

   B. En aquest Model de Domini, cal posar un atribut a Estació per cada llista de persones que hi treballen.

   C. Les associacions "TéFinal" i "TéOrigen" entre les classes conceptuals Viatge i Estació no poden existir simultàniament, ja que són dues associacions entre les dues mateixes classes.

   D. Cal una altra classe conceptual Estat per saber si el Ticket ha estat pagat o no.



Contesta a les següents preguntes:

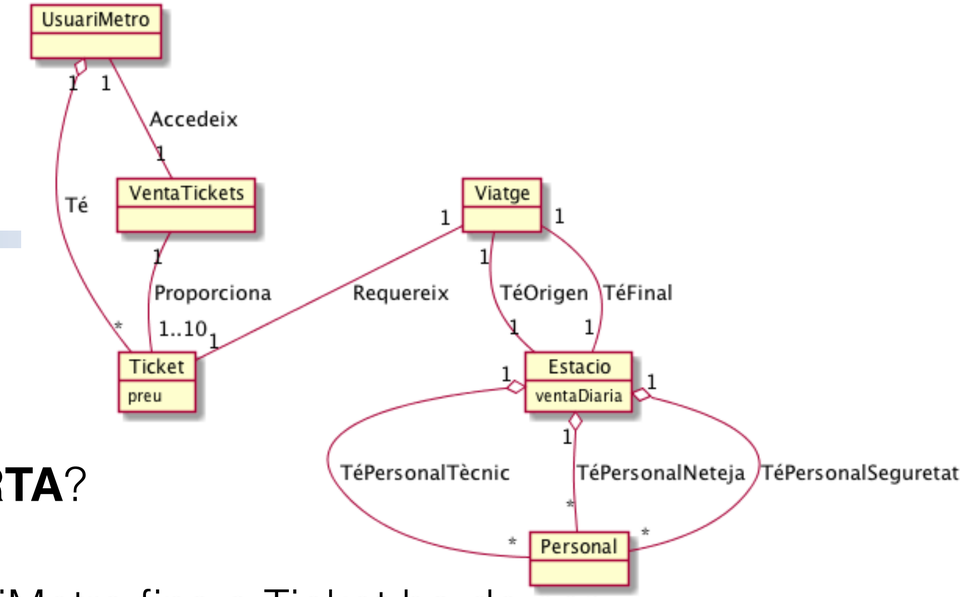
1. Quina de les següents afirmacions és **CERTA**?:

   A. Cal posar Quantitat com tipus de l'atribut "preu" de la classe Ticket per a poder expressar que el preu es pot donar en diferents divises.

   B. En aquest Model de Domini, cal posar un atribut a Estació per cada llista de persones que hi treballen.

   C. Les associacions "TéFinal" i "TéOrigen" entre les classes conceptuals Viatge i Estació no poden existir simultàniament, ja que són dues associacions entre les dues mateixes classes.

   D. Cal una altra classe conceptual Estat per saber si el Ticket ha estat pagat o no.



Contesta a les següents preguntes:

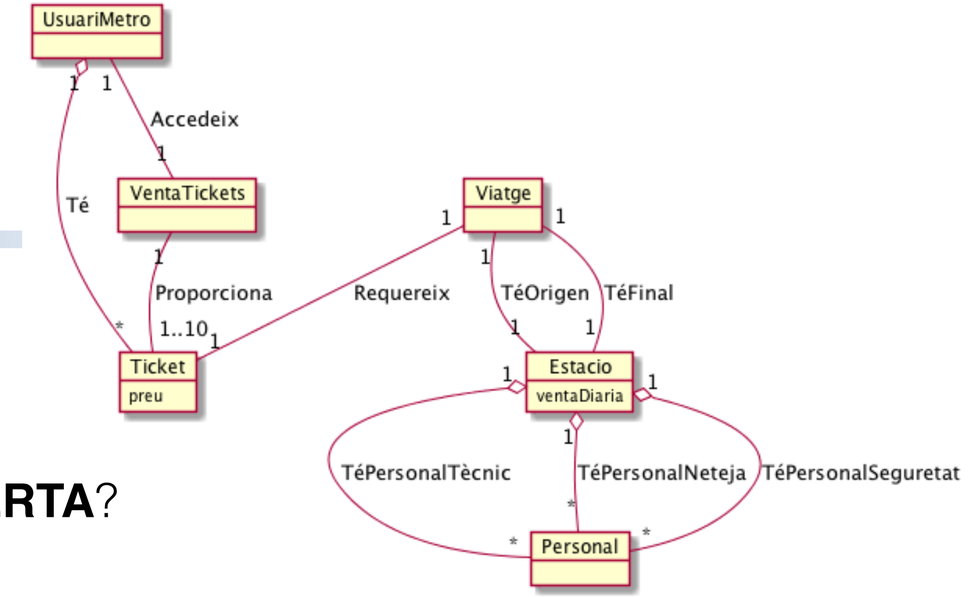
2. Quina de les següents afirmacions és **CERTA**?

A. L'agregació "Té" que va des de l'UsuariMetro fins a Ticket ha de ser una composició i la relació ha de ser 1 -- 1 .. n i no 1 -- *

B. L'associació de VentaTickets amb Ticket ha de ser 1..* i no 1..10 ja que cada usuari pot fer diferents compres de tickets en diferents moments.

C. En lloc de 3 relacions d'Estació a Personal es podria fer una herència de la classe conceptual Personal amb tres classes que especialitzen el Personal, (Tècnic, Neteja i Seguretat) i només fer una associació entre Estació i la classe pare Personal.

D. L'usuari de metro no pot ser una classe conceptual ja que és un actor del sistema.



Contesta a les següents preguntes:

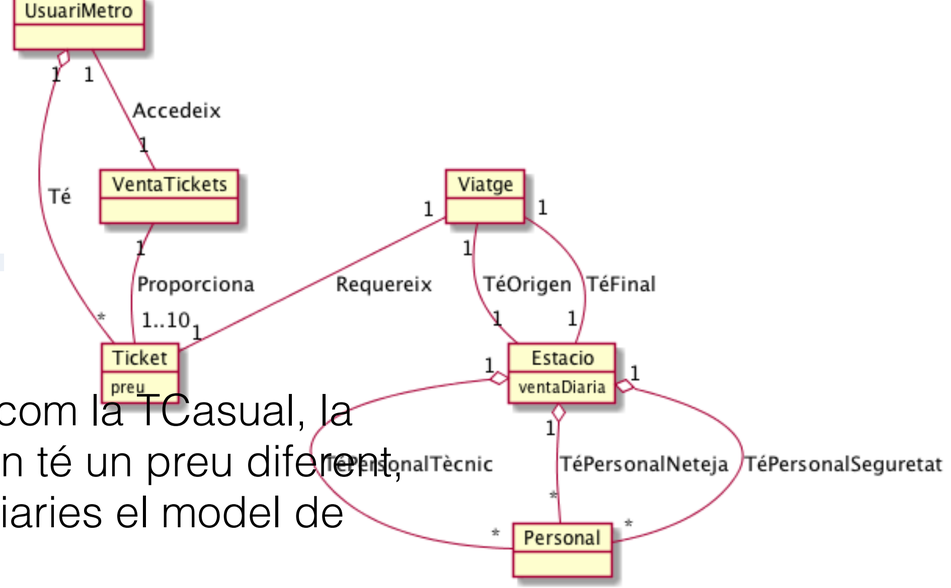
2. Quina de les següents afirmacions és **CERTA**?

A. L'agregació "Té" que va des de l'UsuariMetro fins a Ticket ha de ser una composició i la relació ha de ser 1 -- 1 .. * i no 1 -- *

B. L'associació de VentaTickets amb Ticket ha de ser 1..* i no 1..10 ja que cada usuari pot fer diferents compres de tickets en diferents moments.

C. En lloc de 3 relacions d'Estació a Personal es podria fer una herència de la classe conceptual Personal amb tres classes que especialitzen el Personal, (Tècnic, Neteja i Seguretat) i només fer una associació entre Estació i la classe pare Personal.

D. L'usuari de metro no pot ser una classe conceptual ja que és un actor del sistema.



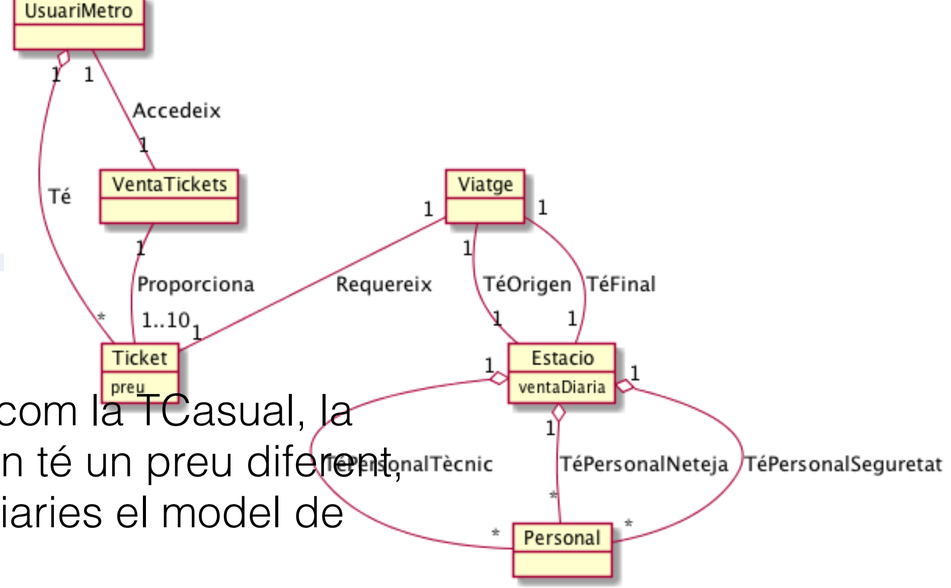
3. Si es volgués tenir diferents tipus de bitllets de metro, com la TCasual, la TUsual, la T50/30, abono mensual, trimestral on cadascun té un preu diferent, amb diferents tipus de condicions d'utilització, com canviaries el model de domini anterior:

A. Afegint un atribut tipusTicket de tipus String en la classe conceptual Ticket.

B. Especialitzant la classe conceptual Ticket en diferents classes filles que corresponen a cadascun del tipus de Ticket. A la classe Ticket es manté l'atribut preu i a cadascuna de les classes filles es tenen com a atributs les seves característiques particulars.

C. Especialitzant la classe conceptual Ticket en diferents classes filles que corresponen a cadascun del tipus de Ticket. A cadascuna de les classes filles es tenen com a atributs el preu i les seves característiques particulars.

D. Cap de les maneres anteriors conceptualitza la situació descrita.



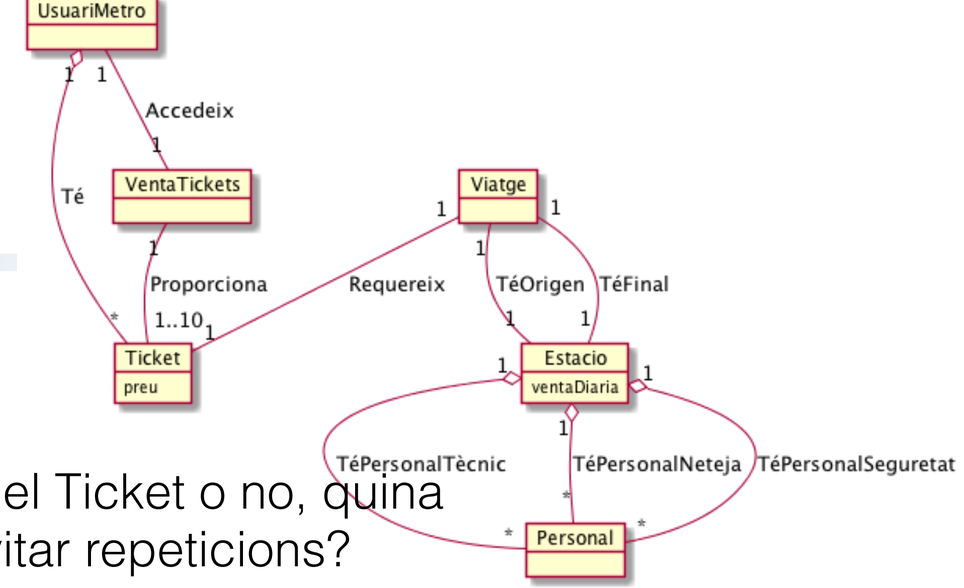
3. Si es volgués tenir diferents tipus de bitllets de metro, com la TCasual, la TUsual, la T50/30, abono mensual, trimestral on cadascun té un preu diferent, amb diferents tipus de condicions d'utilització, com canviaries el model de domini anterior:

A. Afegint un atribut tipusTicket de tipus String en la classe conceptual Ticket.

B. Especialitzant la classe conceptual Ticket en diferents classes filles que corresponen a cadascun del tipus de Ticket. A la classe Ticket es manté l'atribut preu i a cadascuna de les classes filles es tenen com a atributs les seves característiques particulars.

C. Especialitzant la classe conceptual Ticket en diferents classes filles que corresponen a cadascun del tipus de Ticket. A cadascuna de les classes filles es tenen com a atributs el preu i les seves característiques particulars.

D. Cap de les maneres anteriors conceptualitza la situació descrita.



Contesta a les següents preguntes:

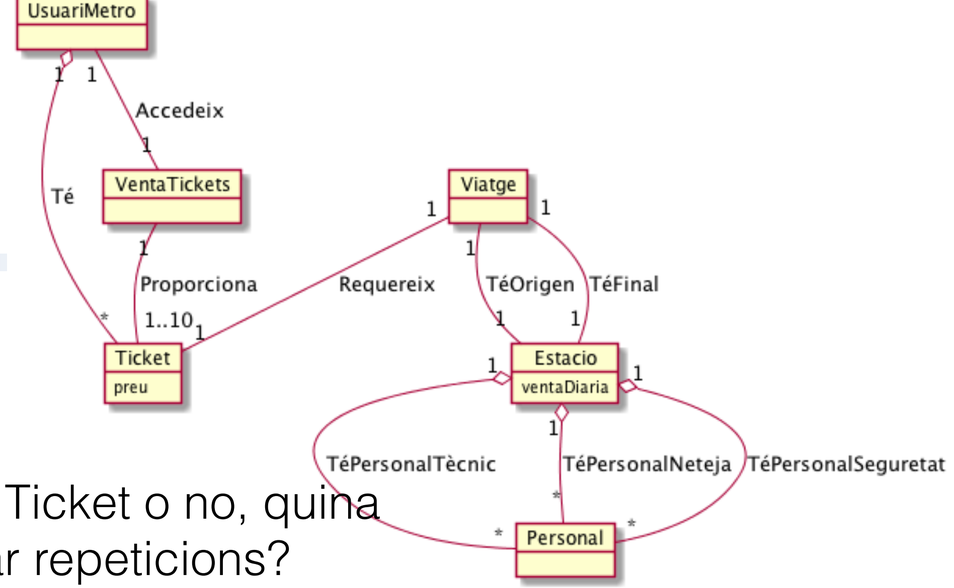
4. En el cas que es volés modelar si s'ha fet servir el Ticket o no, quina modificació modelaria millor aquesta idea per a evitar repeticions?

A. Es té una classe conceptual Estat que té dos fills, TicketNou i TicketUsat. S'inclou una associació entre Ticket i Estat anomenada "Té un" definida com "Ticket "1"-- “1..*” Estat : "Té un".

B. Es té un atribut booleà anomenat "usat" en la classe conceptual Ticket.

C. El Ticket es generalitza en dues filles TicketNou i TicketUsat.

D. Es té una classe conceptual Estat que té dos fills, TicketNou i TicketUsat. S'inclou una associació entre Ticket i Estat anomenada "Té un" definida com "Ticket "*"-- “1" Estat : "Té un”.



Contesta a les següents preguntes:

4. En el cas que es volés modelar si s'ha fet servir el Ticket o no, quina modificació modelaria millor aquesta idea per a evitar repeticions?

A. Es té una classe conceptual Estat que té dos fills, TicketNou i TicketUsat. S'inclou una associació entre Ticket i Estat anomenada "Té un" definida com "Ticket "1"-- “1..*” Estat : "Té un".

B. Es té un atribut booleà anomenat "usat" en la classe conceptual Ticket.

C. El Ticket es generalitza en dues filles TicketNou i TicketUsat.

D. Es té una classe conceptual Estat que té dos fills, TicketNou i TicketUsat. S'inclou una associació entre Ticket i Estat anomenada "Té un" definida com "Ticket "*"-- “1" Estat : "Té un”.

Activitat a classe

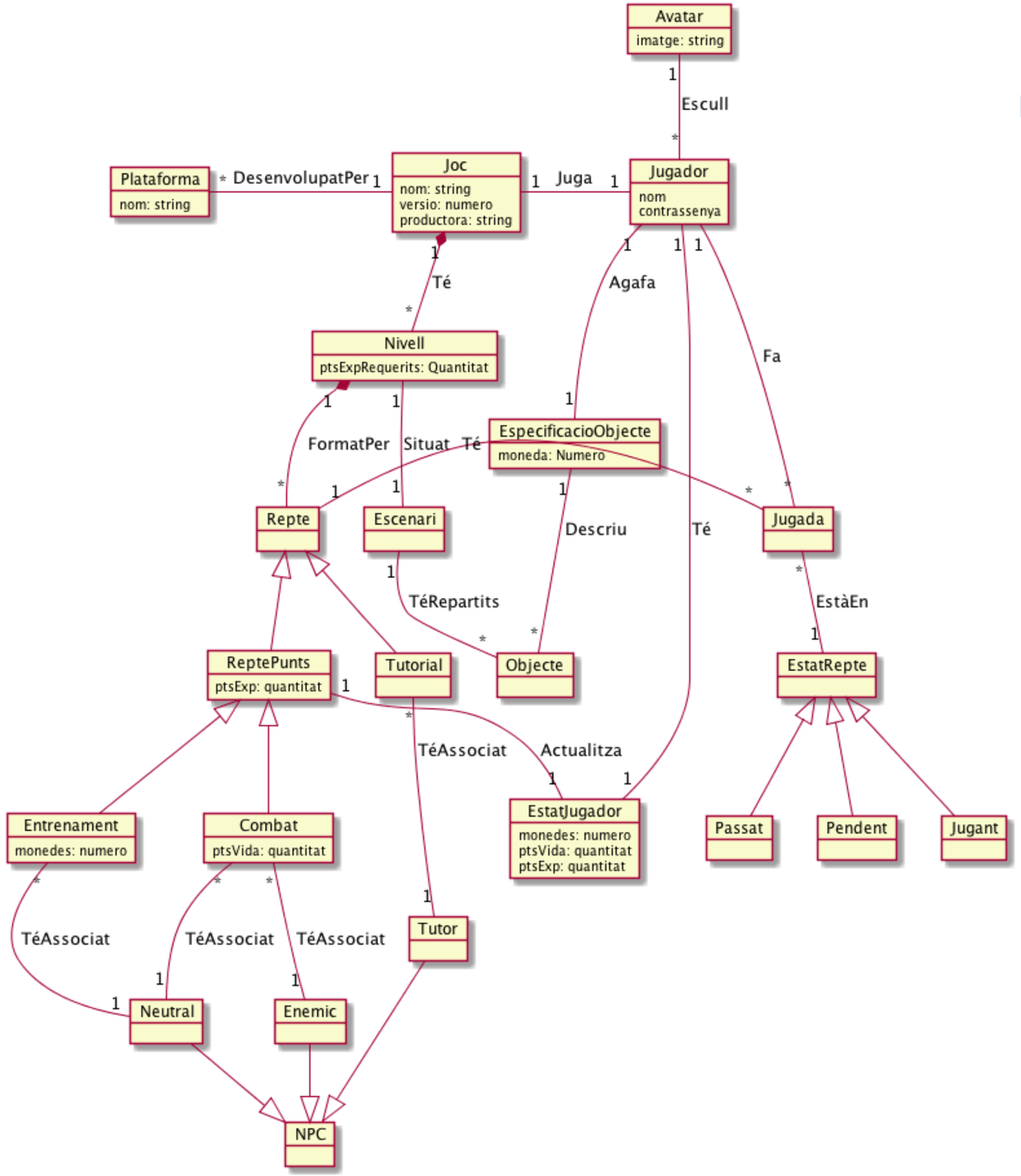
Resoldre examen de Model de Domini (2016-17):

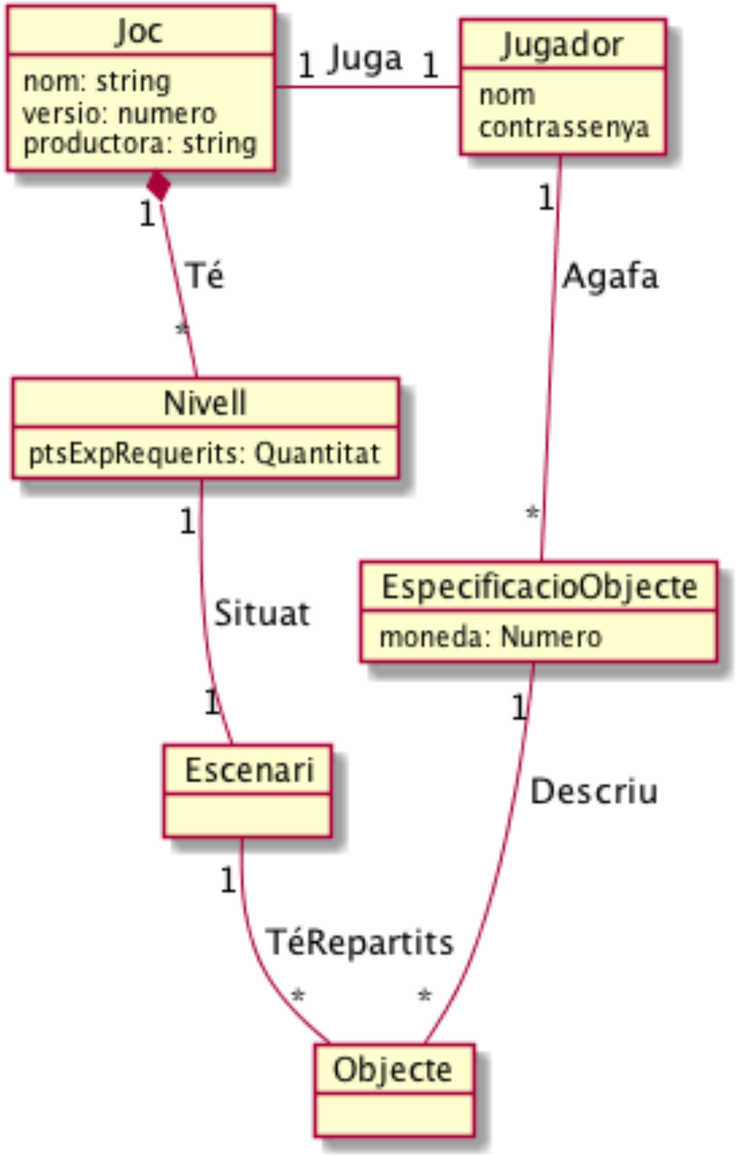
Joc d'aventura mono-jugador:

1. Modelar nivells del joc
2. Modelar un moment de la partida d'un jugador
3. Restricció que el jugador només pugui agafar un objecte només un cop

https://campusvirtual.ub.edu/mod/resource/view.php?id=2668577

MD de JOC


Avatar
imatge: string
Escull
Plataforma
nom: string
DesenvolupatPer
Joc
nom: string
versio: numero
productora: string
Juga
Jugador
nom
contrassenya
Té
Agafa
Fa
Nivell
ptsExpRequerits: Quantitat
FormatPer
Situat
Té
EspecificacioObjecte
moneda: Numero
Repte
Escenari
Descriu
Té
Jugada
TéRepartits
EstàEn
ReptePunts
ptsExp: quantitat
Tutorial
Objecte
EstatRepte
TéAssociat
Actualitza
Entrenament
monedes: numero
Combat
ptsVida: quantitat
EstatJugador
monedes: numero
ptsVida: quantitat
ptsExp: quantitat
Passat
Pendent
Jugant
TéAssociat
TéAssociat
TéAssociat
Tutor
Neutral
Enemic
NPC


Joc
nom: string
versio: numero
productora: string
1 Juga 1
Jugador
nom
contrassenya
1
Té
*
1
Agafa
*
Nivell
ptsExpRequerits: Quantitat
1
Situat
1
EspecificacioObjecte
moneda: Numero
1
Descriu
Escenari
1
TéRepartits
* *
Objecte

Activitat a classe

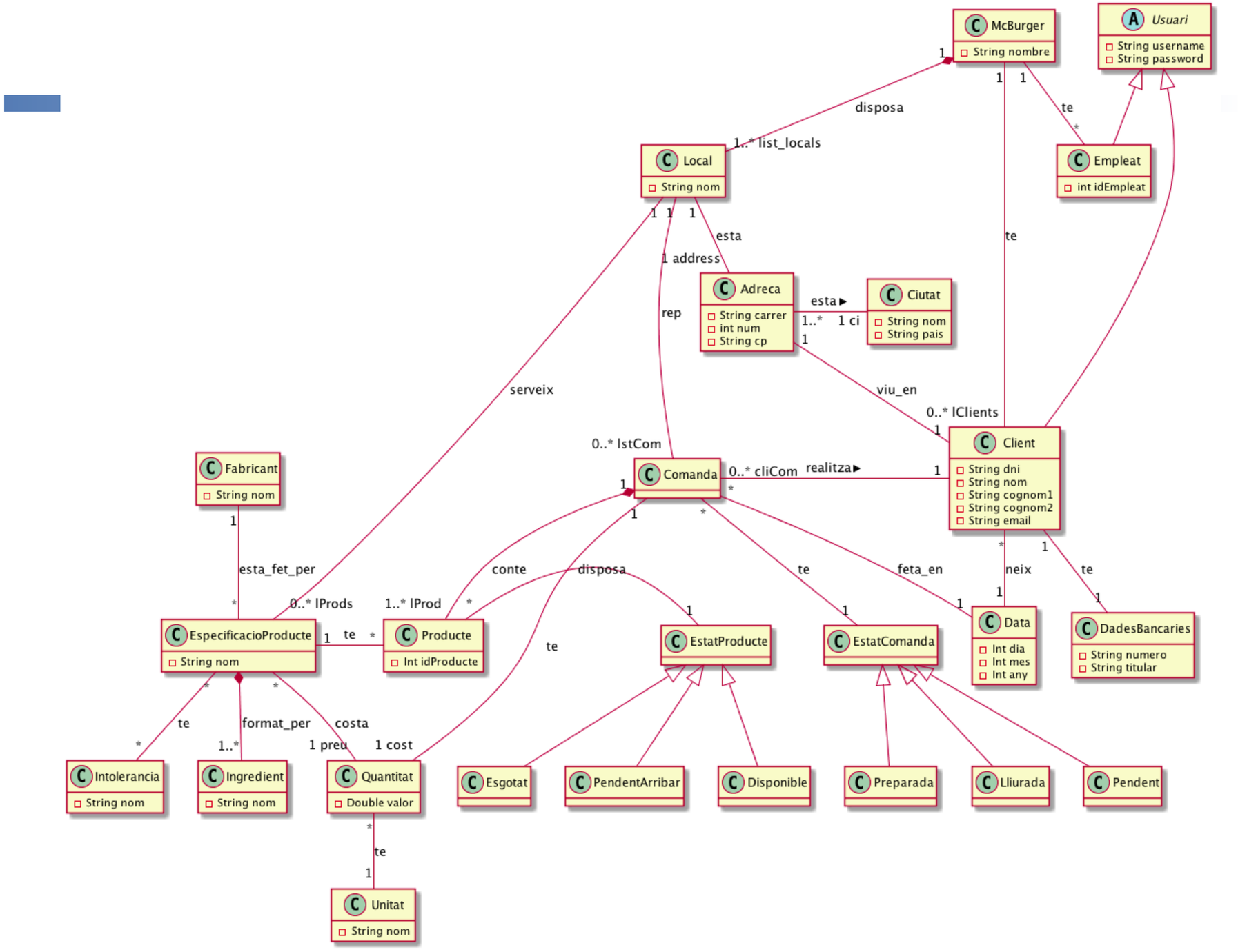
Resoldre examen de Model de Domini (2020-21):

EMPRESA MCBURGUER:

1. Entendre el Model de Domini de l'examen
2. Modificar el Model de Domini: planteja't calsses d'especificació, d'estat, etc.

https://campusvirtual.ub.edu/mod/resource/view.php?id=3057425

Model de domini de McBurger



Temari

1	Introducció al procés de desenvolupament del software
2	Anàlisi de requisits i especificació
3	Disseny
4	Del disseny a la implementació
5	Ús de frameworks de testing

3.1	**Introducció**
3.2	Patrons arquitectònics
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

Temari (vídeos)

Introducció al disseny (metodologia i patrons)

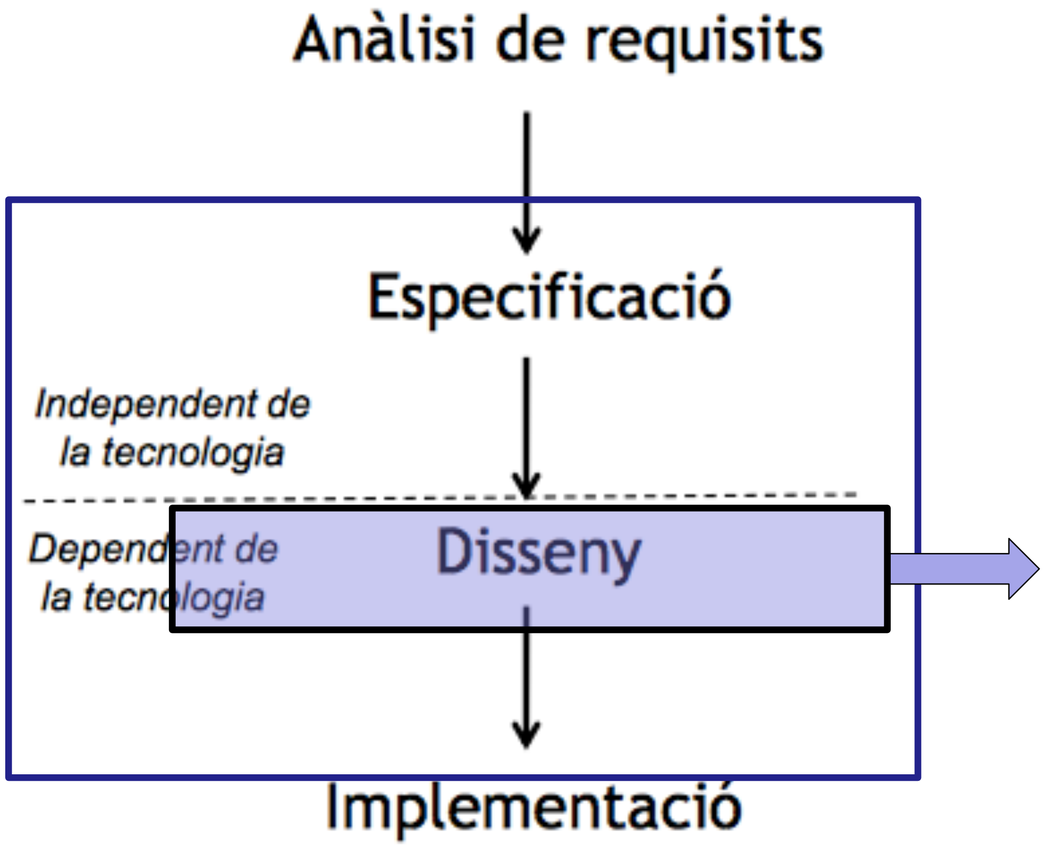
Exemple de la Metodologia a seguir

Propietats del paradigma Orientat a Objectes

Transparències de suport Tema 3.1

3.1. Introducció

Procés sistemàtic (Tema 3):

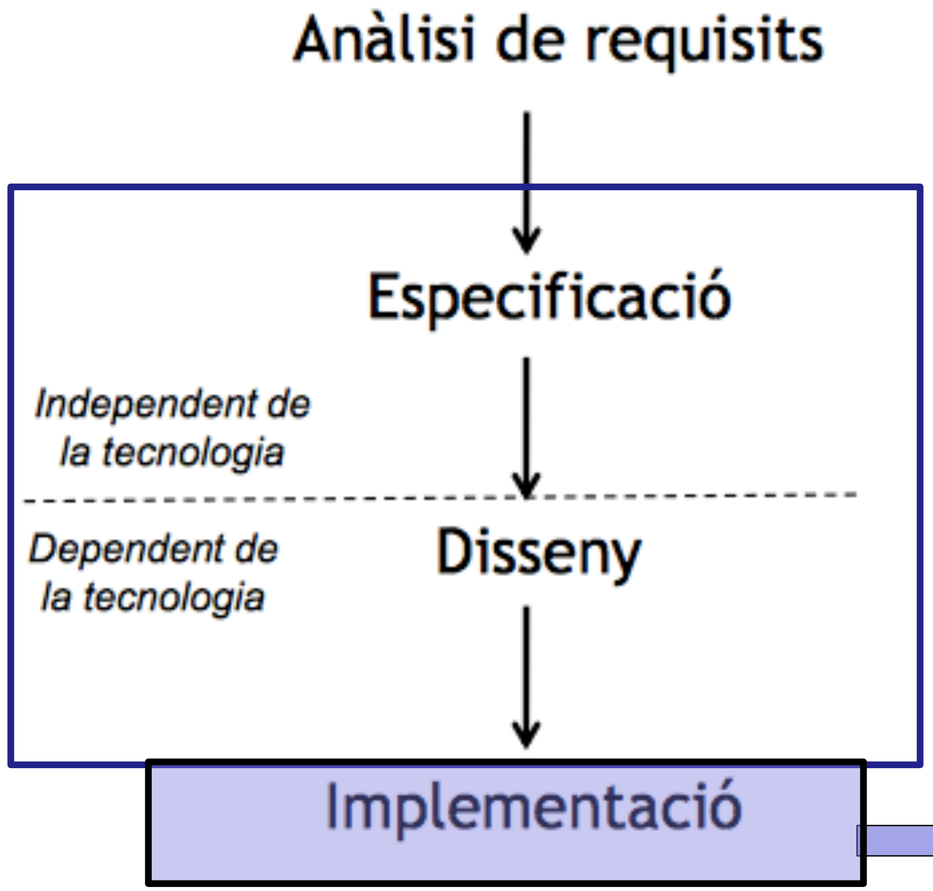


Com ho fa el sistema?

- **Tests d'acceptació**
- **Descripció** de l'**arquitectura** dels subsistemes i components del sistema software
- **Diagrama de classes**
- **Diagrames d'interacció**

3.1. Introducció

Procés sistemàtic (Laboratori):



Generació de Codi:

- **Tests unitaris**
- **Codi en llenguatge concret de les classes**

3.1. Introducció

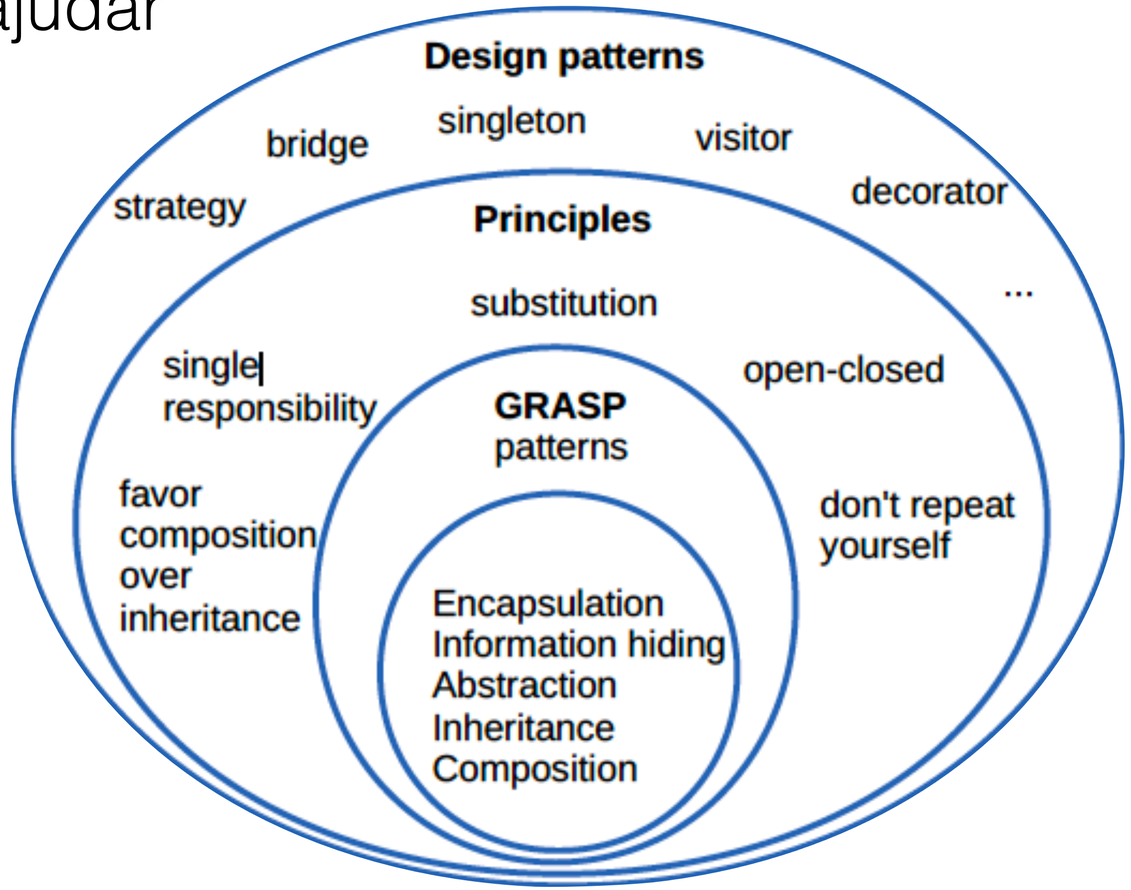
Per a què es vol un **bon** disseny de software?

- 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**)

Com disseny en OO?

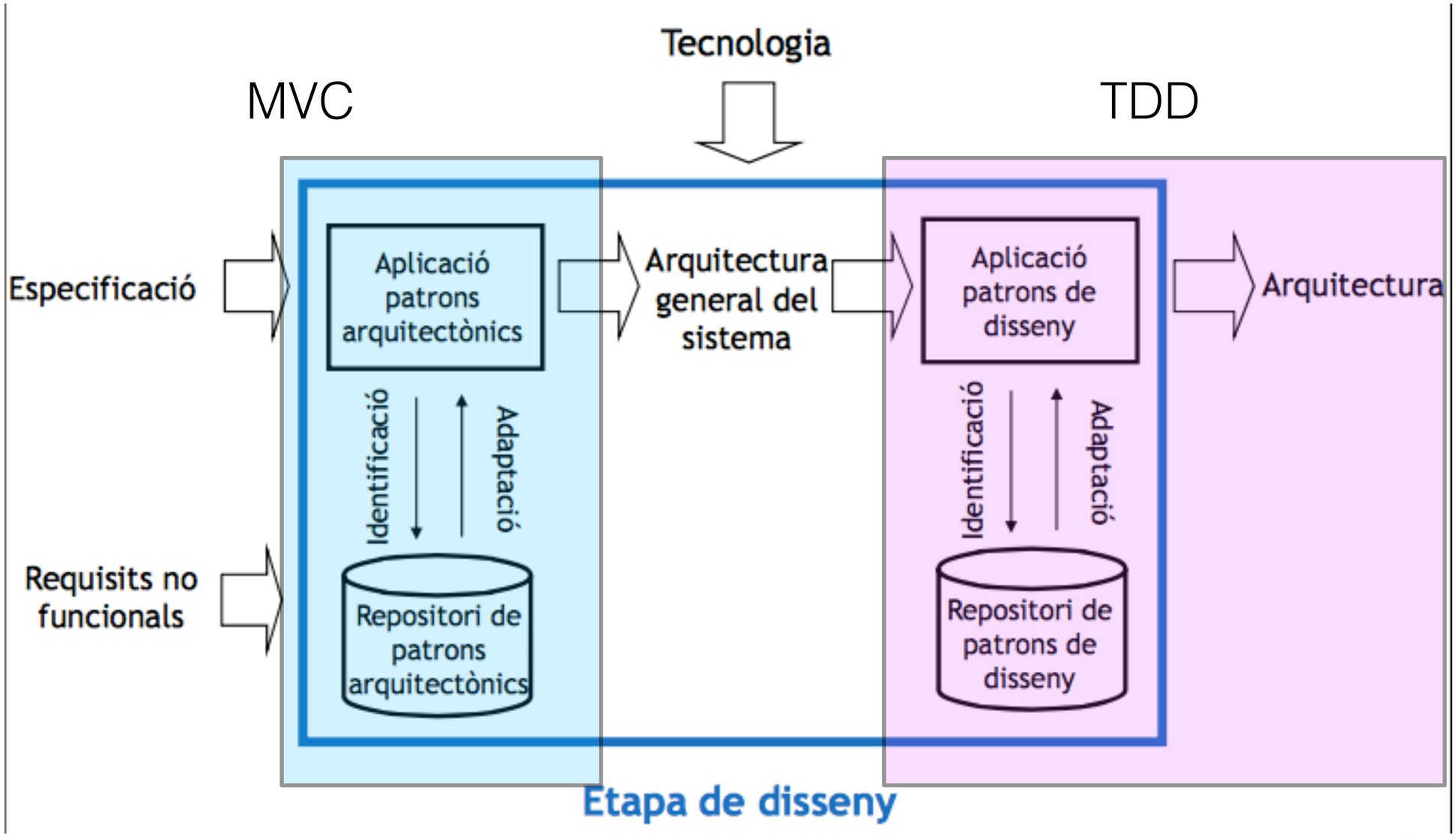
- No hi ha una metodologia que doni el millor disseny però hi han principis, heurístiques i patrons que poden ajudar



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

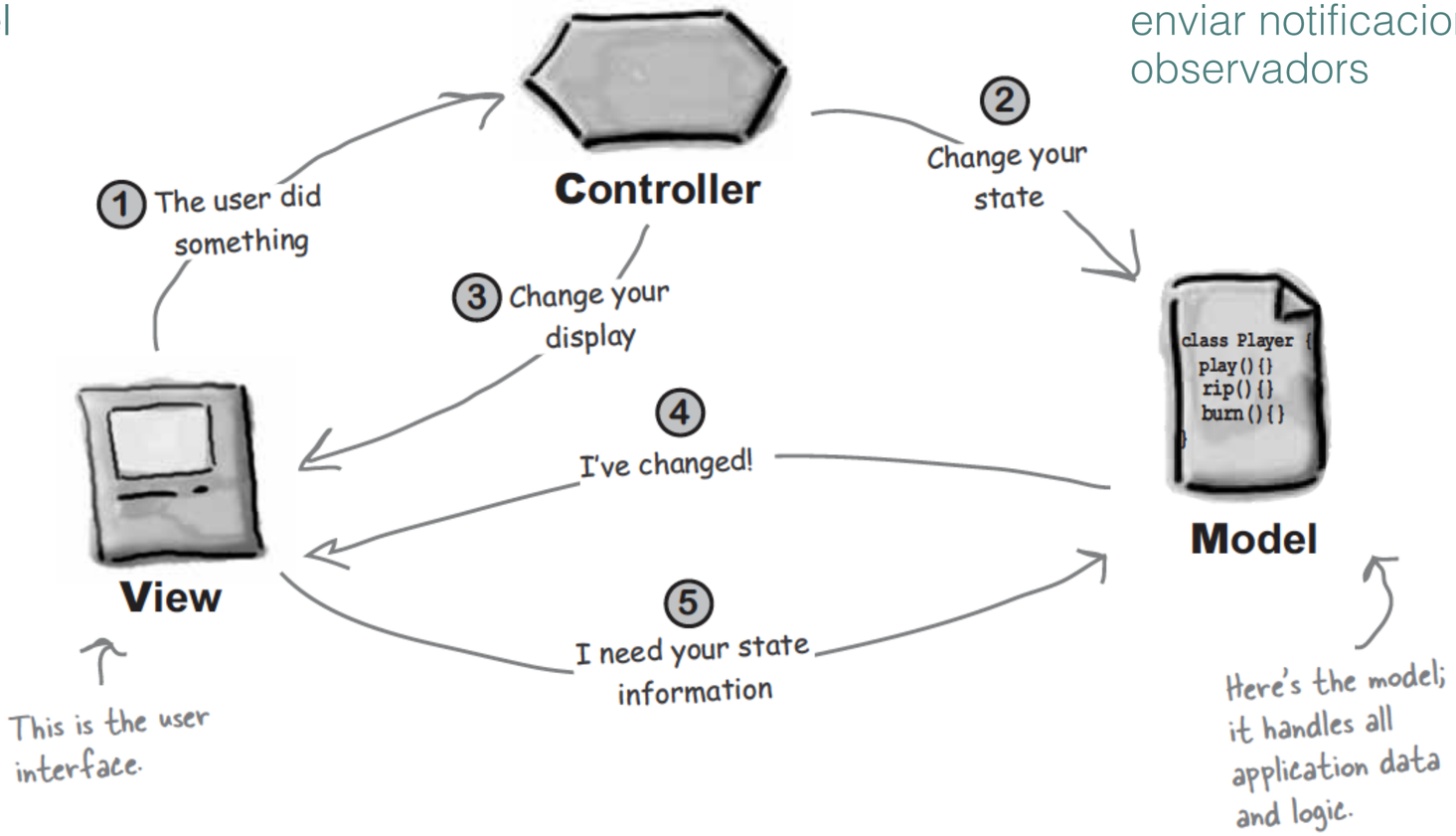


3.1. Introducció

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

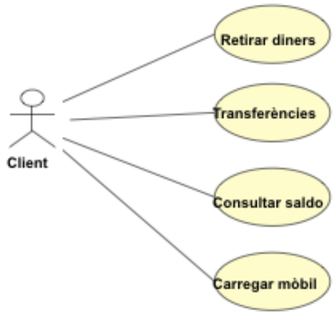


3.1. Introducció

Escenario 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.
...

Casos d'ús

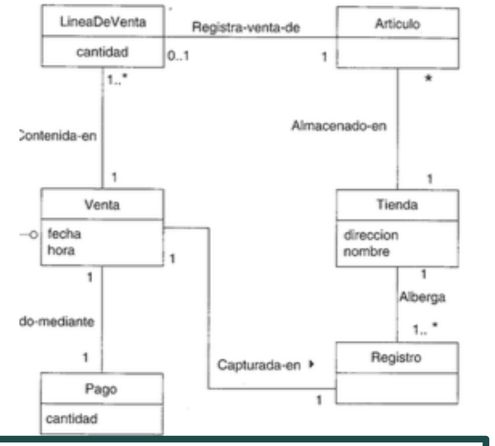


DCU

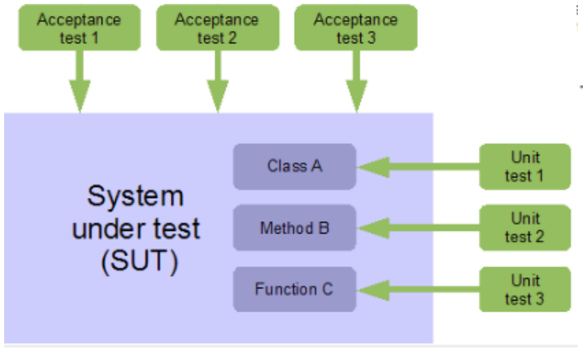
Com a [rol d'usuari]
vull [objectiu]
per què així [raó]

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

User Stories+Criteris d'acceptació



Model de Domini



Tests d'acceptacio + Tests unitaris

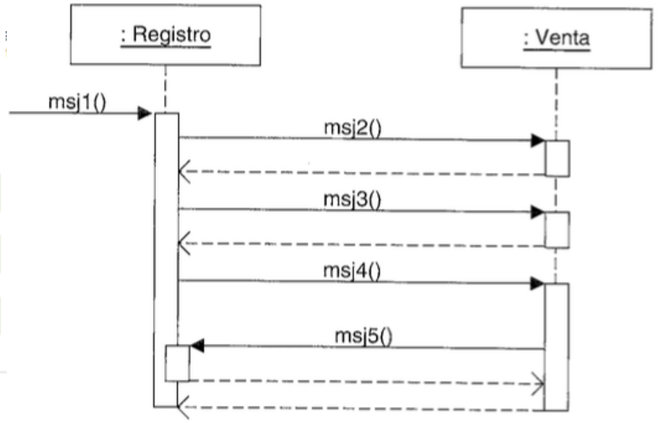


Diagrama de Seqüència (DS)

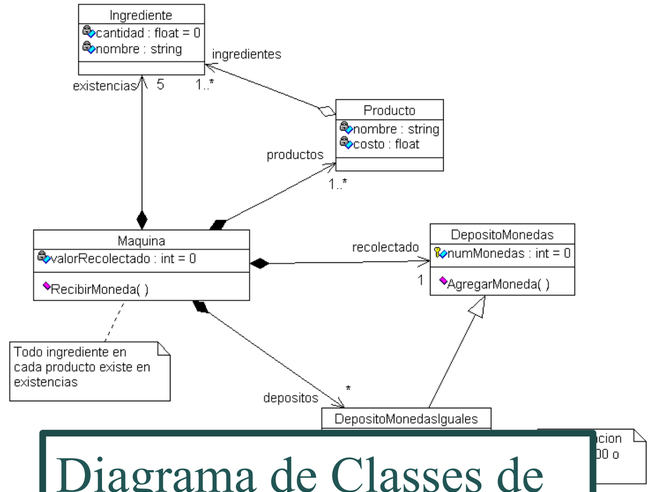
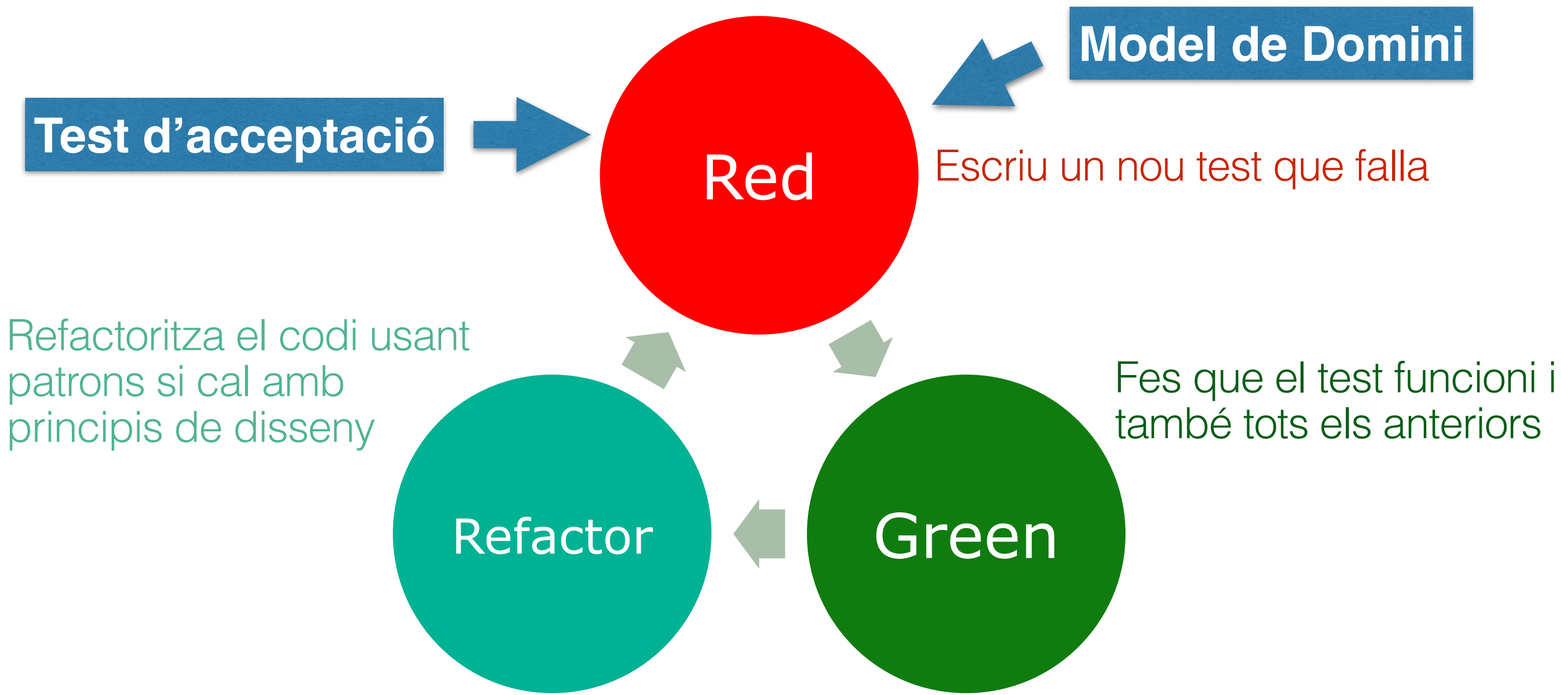


Diagrama de Classes de Disseny (DCD)

3.1. Introducció

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

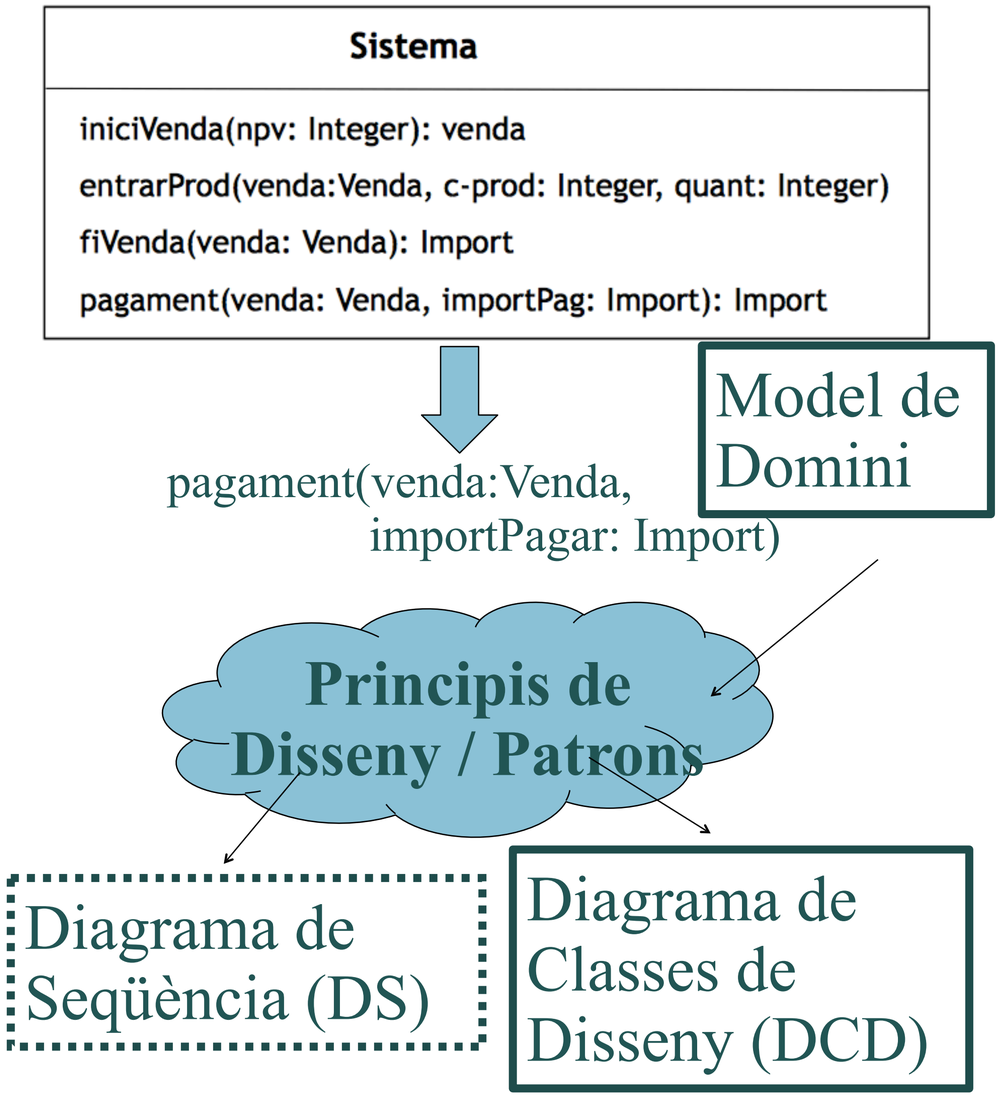
1. Escriu el nou codi només si el test automàtic ha fallat (**Disseny** i **Implementació**)
2. Elimina la duplicació de codi. (**Disseny**)



Passos en el disseny

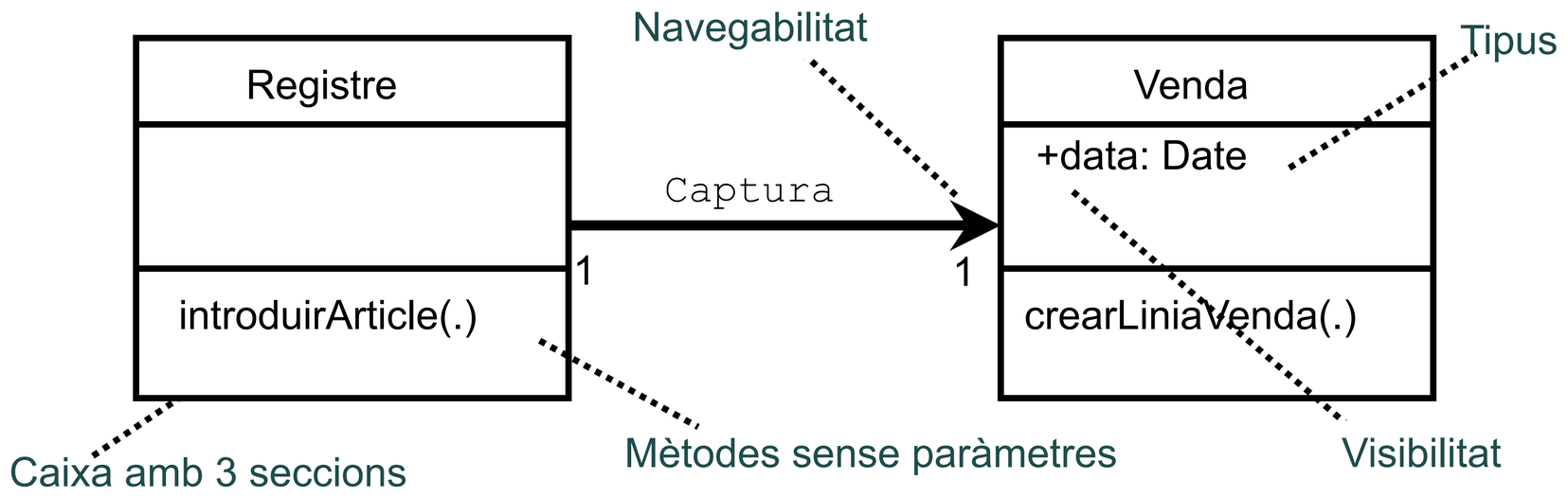
Per a cada **test d'acceptació** definit a l'especificació

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 conceptuals 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



Visibilitat

Visibilitat? defineix quins objectes tenen dret a consultar i eventualment modificar informació declarada en un diagrama de classes de disseny

En UML, la ***visibilitat*** s'estableix sobre:

- Atributs d'una classe
- Operacions d'una classe
- Rols d'associacions accessibles des d'una classe (claus foranes)

En UML, l'element x definit a la classe C pot ser:

- *Public* (+): qualsevol classe que veu C, veu x
- *Private* (-): x només és visible des de C
- *Protected* (#): x és visible des de C i des de tots els descendents de C (si n'hi ha)

Àmbit

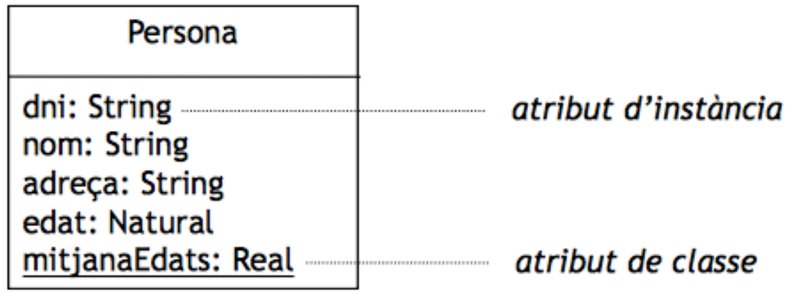
- L'**àmbit** defineix si determinats elements de disseny són aplicables a objectes individuals o a la classe que defineix els elements

En UML, l'àmbit s'estableix sobre:

- Atributs d'una classe
- Operacions d'una classe

En UML, l'element x definit a la classe C pot ser d'àmbit:

- **D'instància** (no estàtic): x està associat als objectes de C referència: obj.x, essent obj una instància de la classe C
- **De classe** (estàtic): x està associat a C referència: C.x



1.4 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"

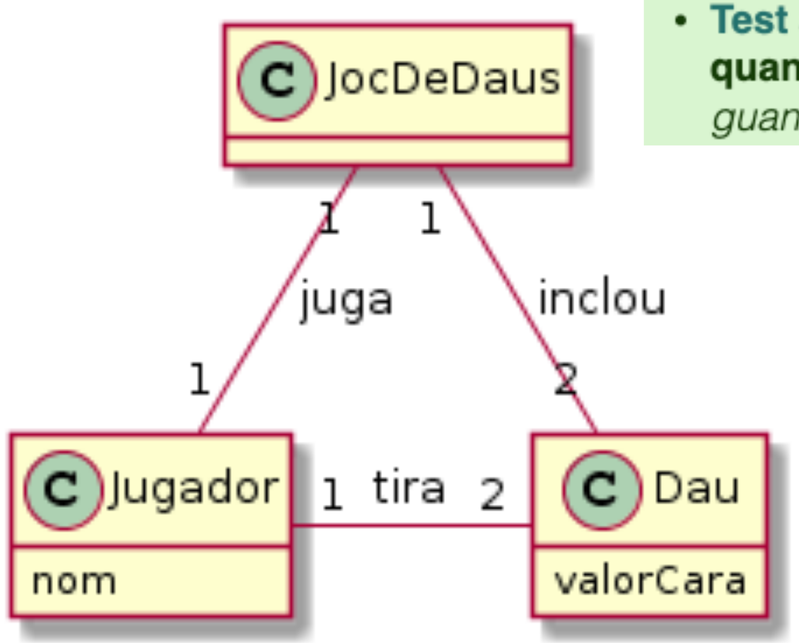
User stories

- **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 perdut"*
- **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"*



Diagrames d'interacció

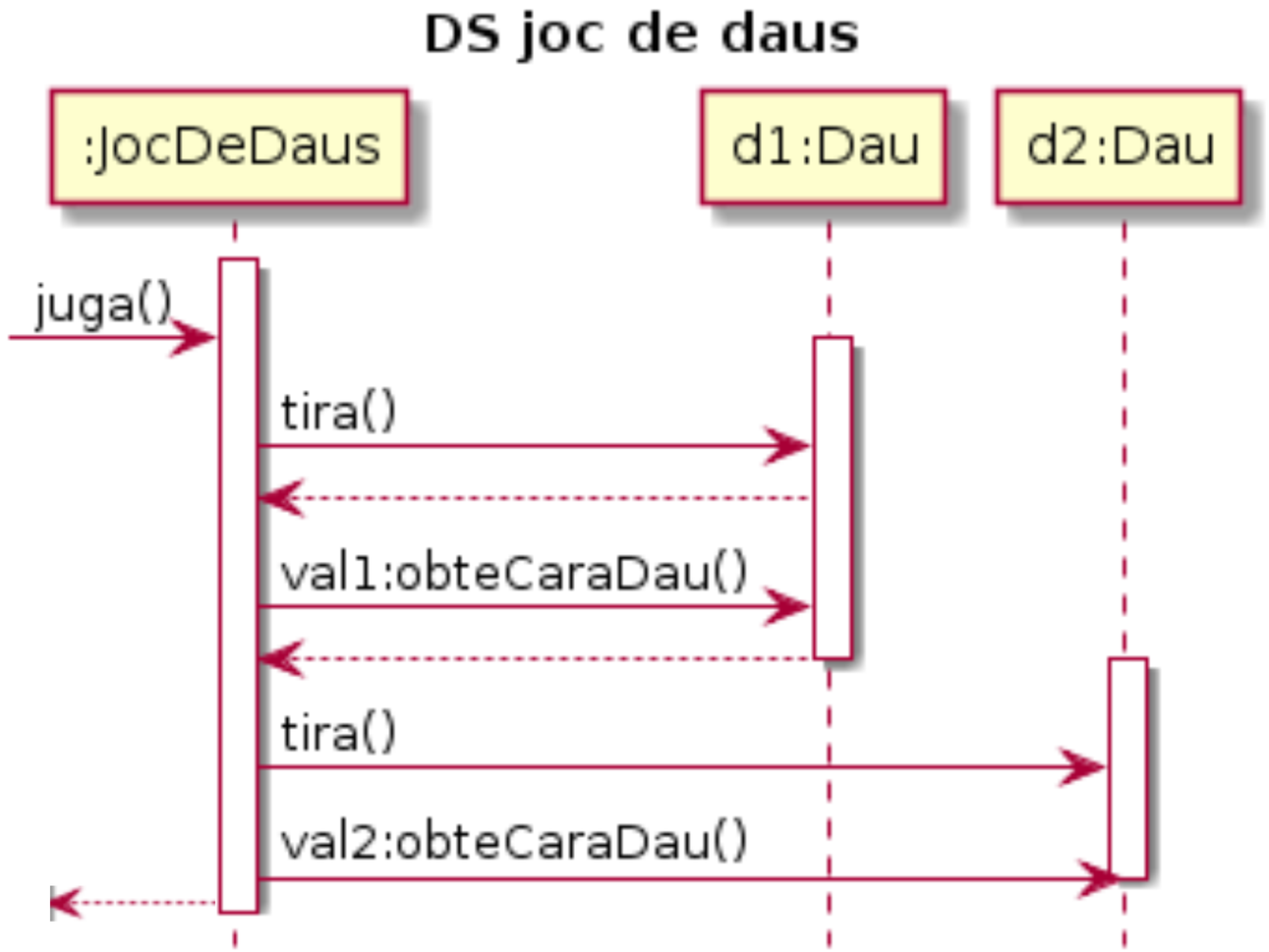
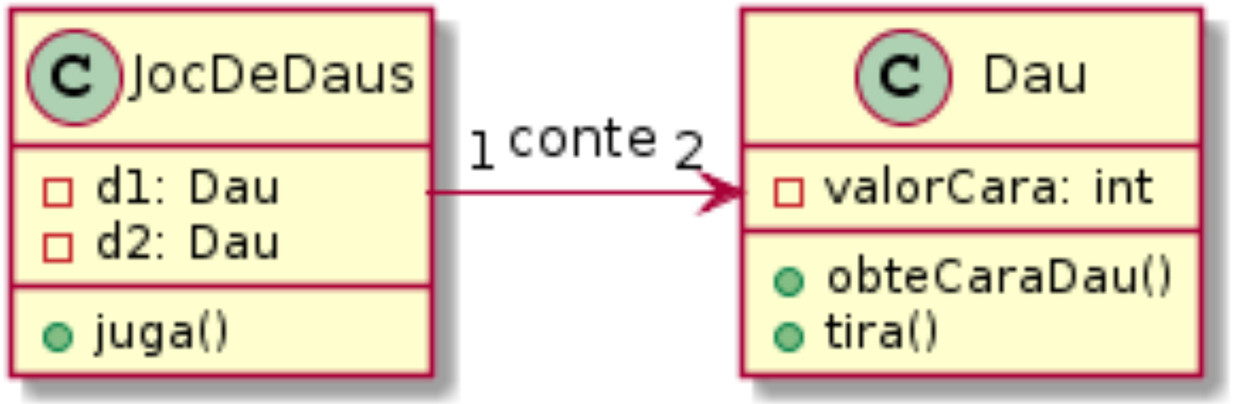


Diagrama de Classes de Disseny



Passos per crear el Diagrama de Classes

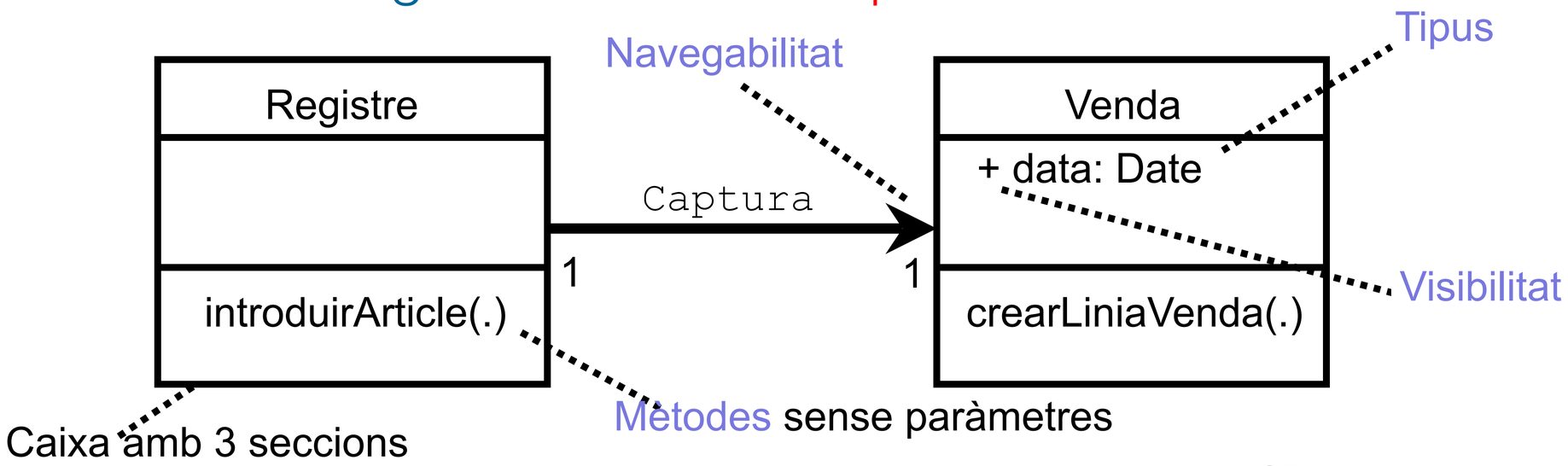
Pas 1. Identificar classes i il·lustrar-les

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



Nova User Story

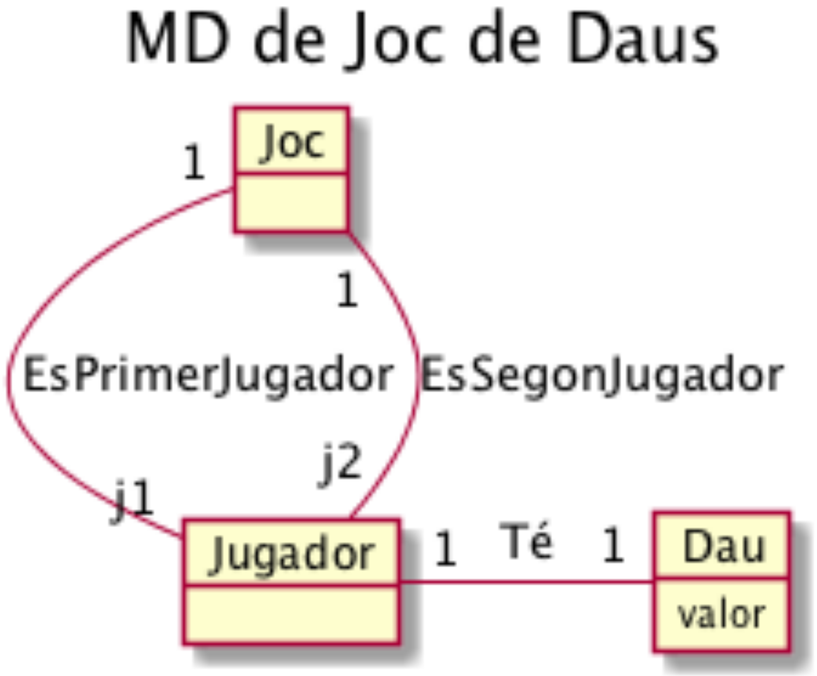
- **Com a** jugador en joc per parelles **vull** jugar amb un altre jugador-me **per a** veure qui treu el dau més alt

- **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



- **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"*

Activitat: Obriu el IntelliJ i feu el codi associat a aquest test d'acceptació

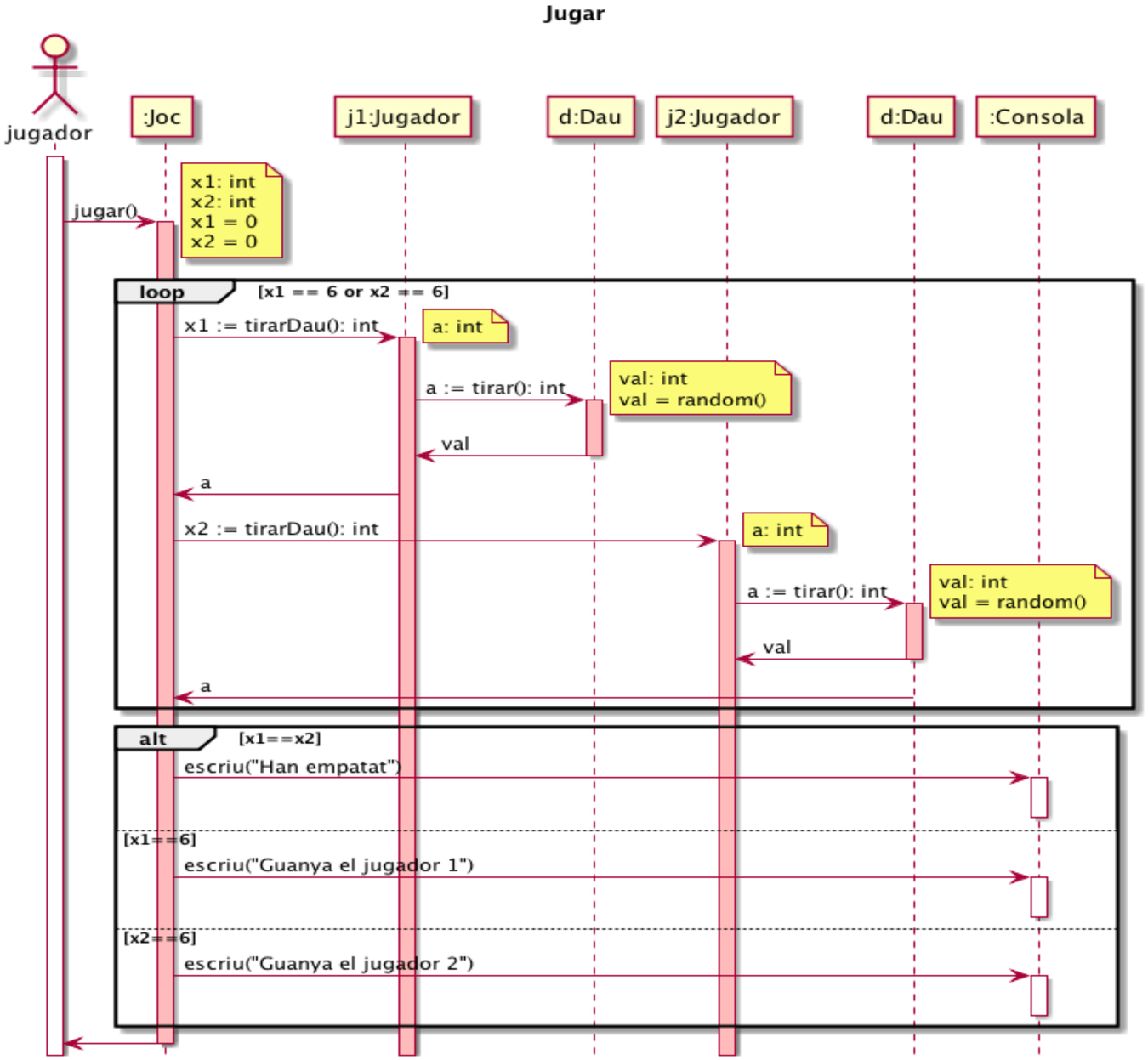
Pas 1. Identificar classes

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

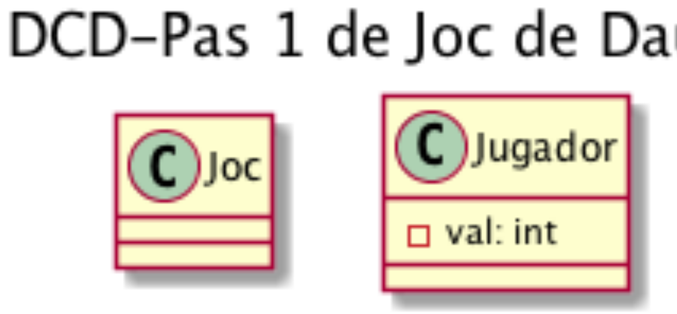
ⓘ No totes les classes de domini es convertiran en classes de disseny

Exemple Joc de Daus



Exemple

Classes i atributs identificades en el Joc de Daus



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

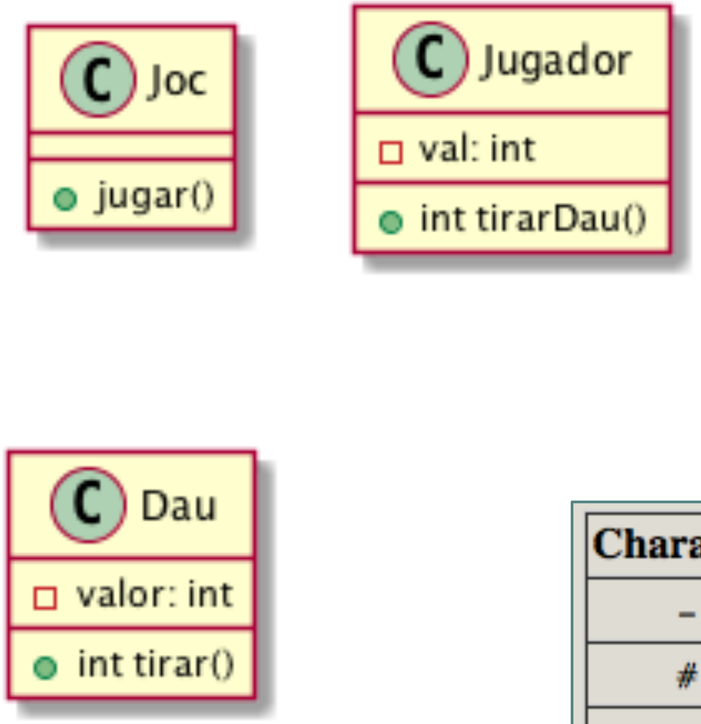
- Si el diagrama es crea per a una eina CASE (com EclipseUML, Rational Rose, ArgoUML o Skecth it! de IntelliJ) amb generació automàtica de codi, s'han de repassar de forma exhaustiva tots els detalls

- 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 associada 1

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

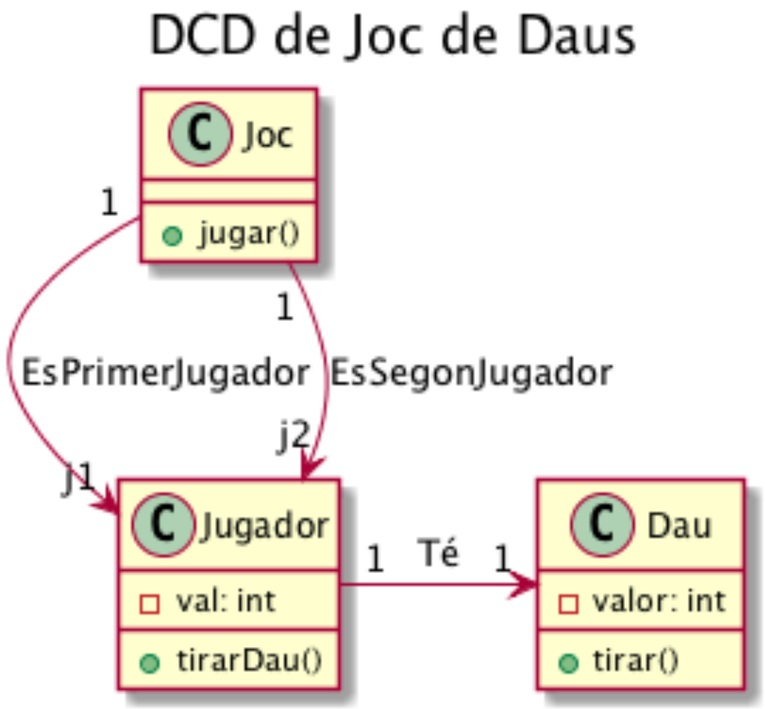
- **Navegabilitat**: és una propietat d'un rol d'una associació que ens indica que és possible navegar unidireccionalment d'objectes de l'origen a objectes del destí segons la direcció indicada per la fletxa

- 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

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
```

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.

3.1. Introducció

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

- **Rigidesa**: l’impacte d’un canvi en el software és impredecible (cada canvi produeix 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 conceptuals
- **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

3.1. Introducció

Característiques que permeten avaluar si un disseny és “bo”:

- **Acoblament:**

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

Acoblament BAIX

Quan més acoblament té una classe:

- més difícil resulta comprendre-la aïlladament.
- més difícil de reutilitzar-la, perquè requereix la presència de les altres classes.

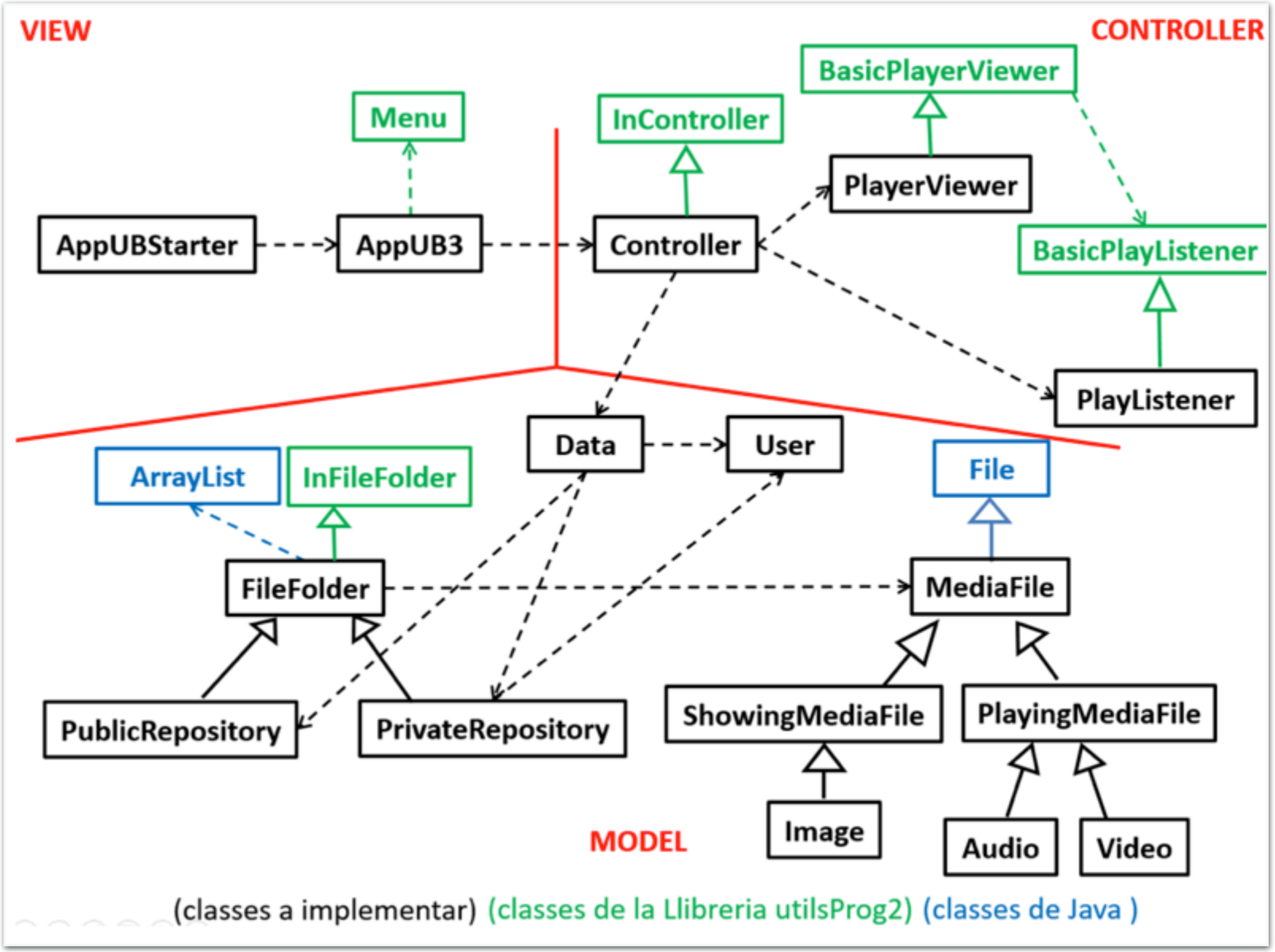
- **Cohesió:**

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

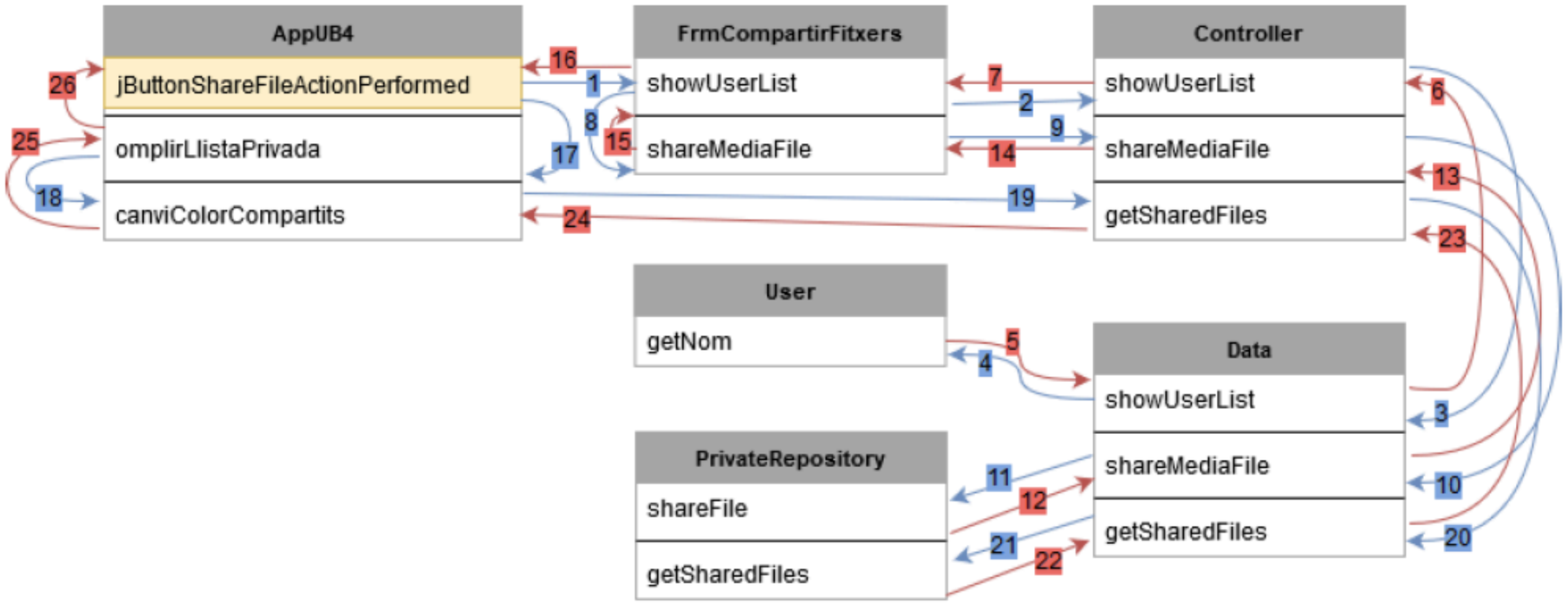
ALTA Cohesió

Una classe amb alta cohesió **no** té molts mètodes, que estan molt relacionats funcionament, i **no** realitza molt de treball. Col·labora amb altres classes.

MVC Prog II


VIEW
CONTROLLER
BasicPlayerViewer
Menu
InController
PlayerViewer
AppUBStarter
AppUB3
Controller
BasicPlayListener
PlayListener
Data
User
ArrayList
InFileFolder
File
FileFolder
MediaFile
PublicRepository
PrivateRepository
ShowingMediaFile
PlayingMediaFile
MODEL
Image
Audio
Video
(classes a implementar) (classes de la Llibreria utilsProg2) (classes de Java)

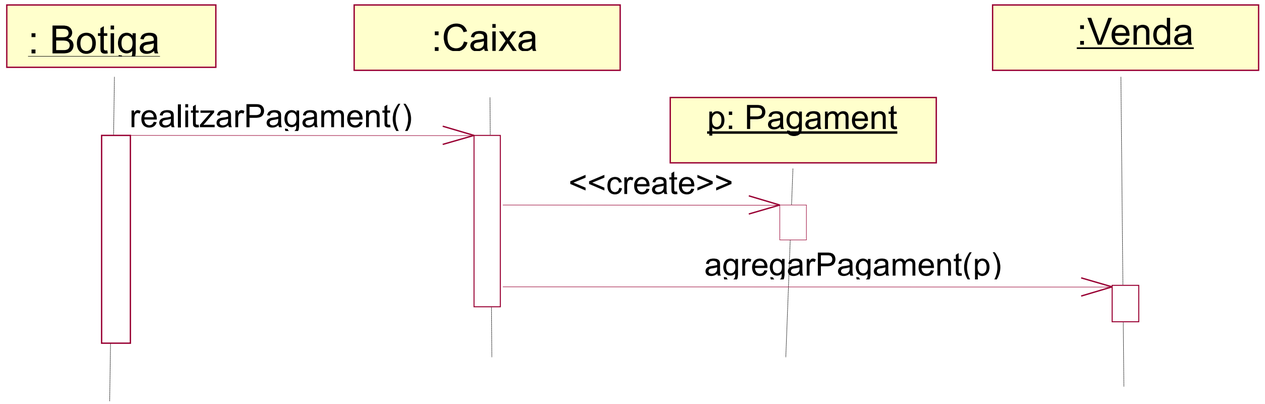
Acoblament? Cohesió?

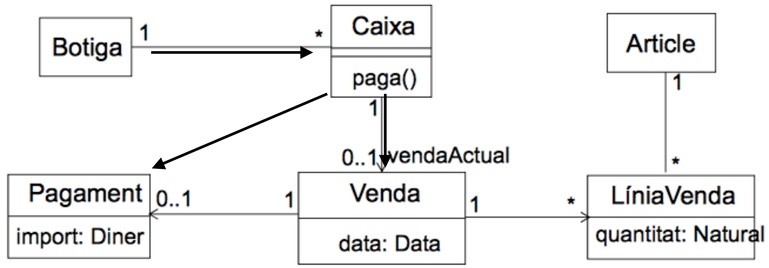


Exemple d'aplicació

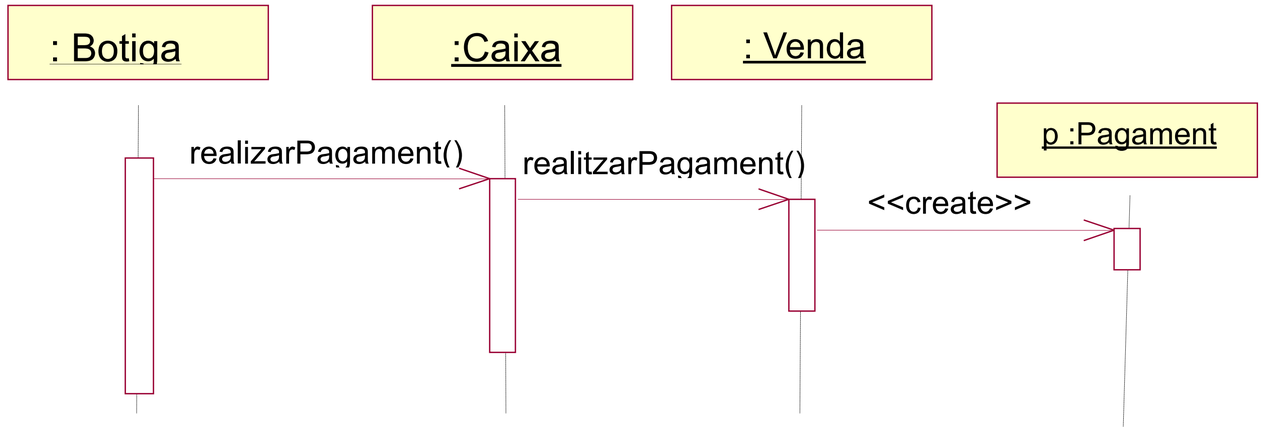
- Què podem dir d'aquests dissenys?

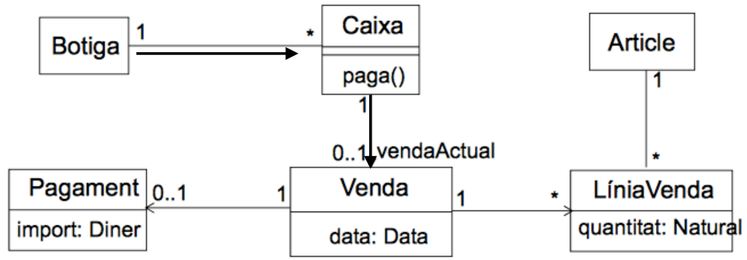
(a)





(b)





Exercici per pensar

- Com aniries construint el Diagrama de Classes i el codi de les funcionalitats de UBCultura?

Objectiu: Partint del Model de Domini, quin Diagrama de Classes obtens després de dissenyar/implementar les següents funcionalitats de test? Detalla els 3 Models de Dominis que vas obtenint…

1. BuscarClient(DNI) retorna el nom del client
2. Afegir un préstec a un client, donat el DNI del client
3. Llistar tot el material en préstec d'UBCultura en un cert dia