



Tema 3: Disseny: Patrons GRASP

Anna Puig

Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Curs 2020/2021

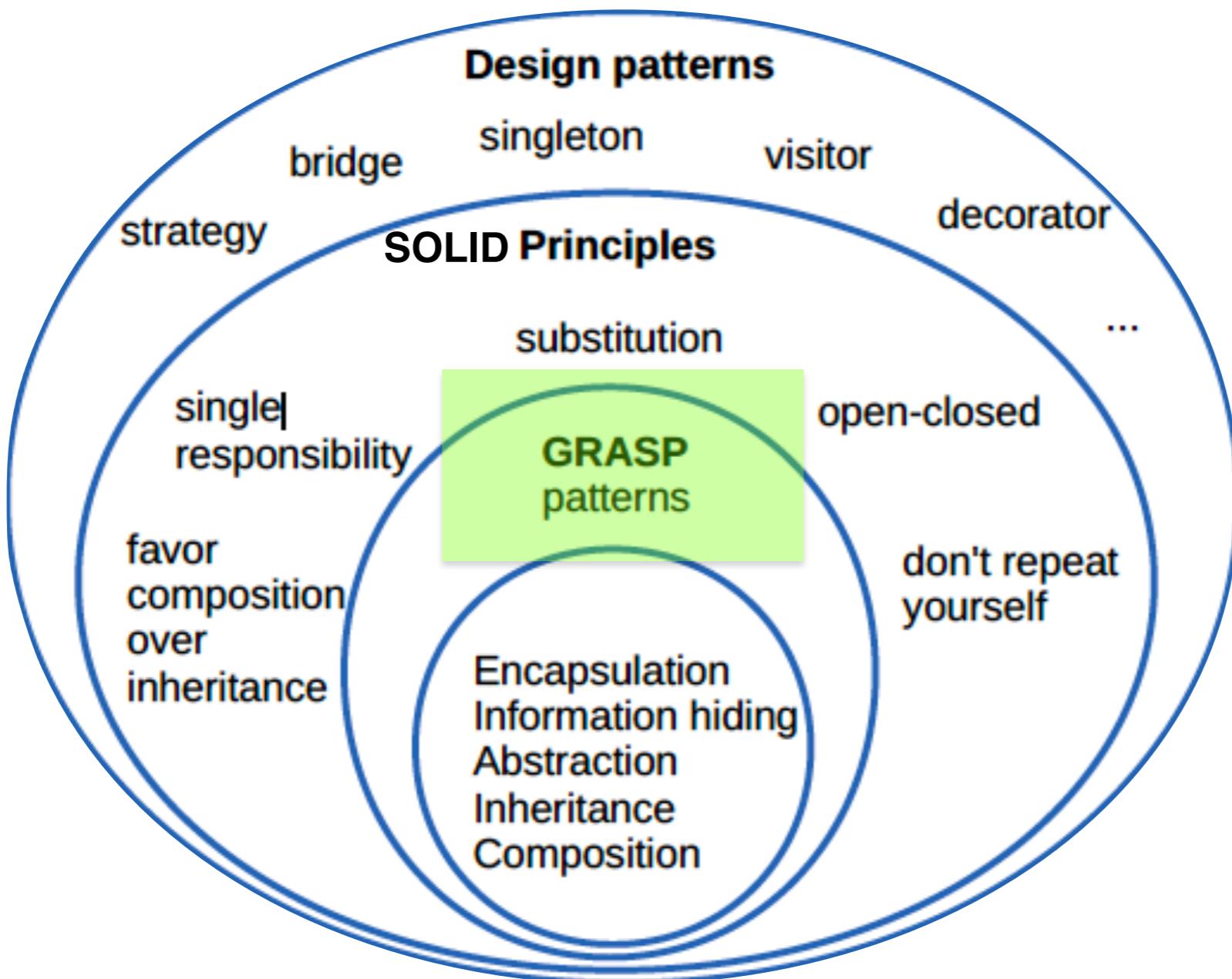
Temari

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

3.3. Criteris GRASP

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



3.3. Criteris GRASP

Per a cada **criteri d'acceptació** identificat en l'especificació es defineix un **test d'acceptació**

1. Es dissenya la seqüència de crides que produirà **l'event** navegant per les classes necessàries
2. A mesura que es necessiten classes s'afegeixen en el **Diagrama de Classes** definint la **navegabilitat** entre elles a partir, si és possible de les classes conceptuais del **Model de Domini**

Com es distribueix el codi per les classes per a resistir **canvis futurs?** Usant els patrons GRASP

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

Model de
Domini



test d'acceptació
+
codi nou (si és necessari)
+
refactoring

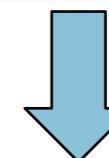
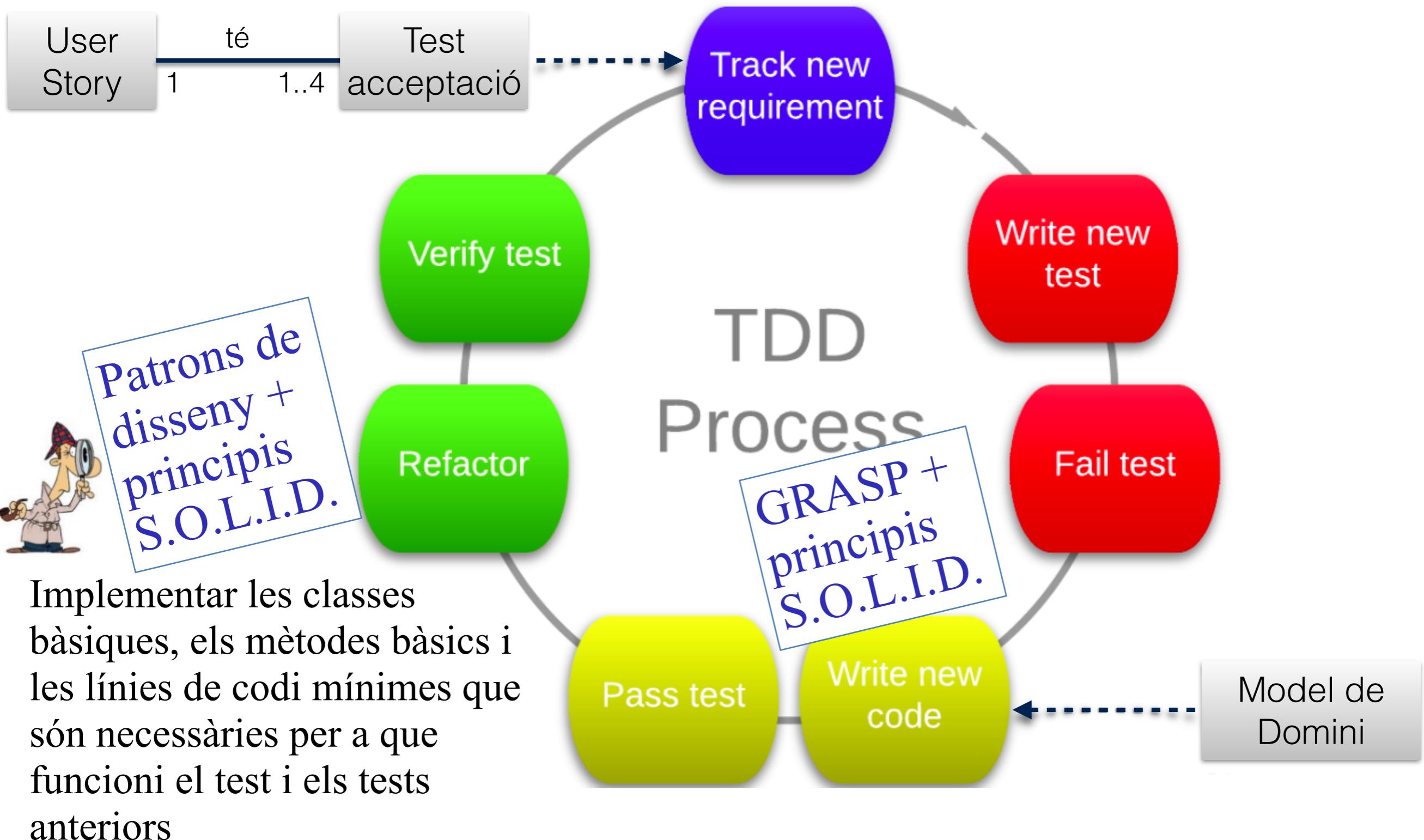


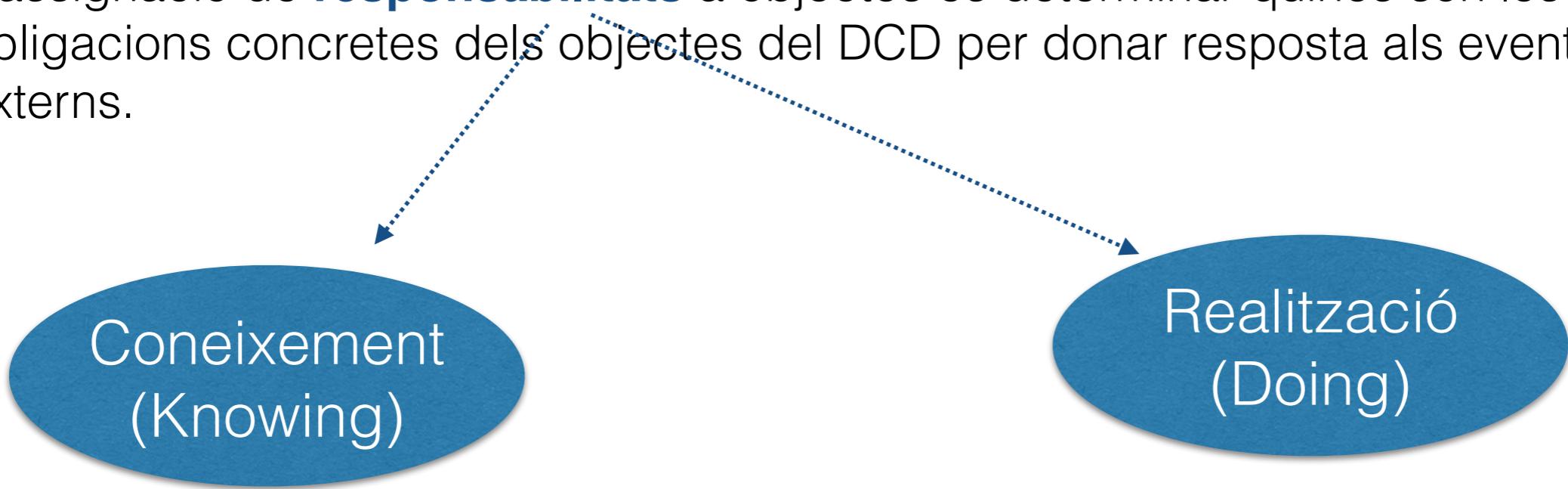
Diagrama de
Classes de
Disseny (DCD)

Quan s'apliquen?



3.1. Criteris GRASP

- **Principis generals (GRASP)**: descripció dels principis bàsics d'assignació de **responsabilitats** a les classes expressades com a patrons
- L'assignació de **responsabilitats** a objectes és determinar quines són les obligacions concretes dels objectes del DCD per donar resposta als events externs.



- Saber sobre els **atributs** privats de l'objecte
- Saber sobre els **objectes** associats
- Saber dades que se'n poden **derivar o calcular**

- Fer alguna cosa en el propi objecte (com crear o fer càculs).
- Iniciar una acció en altres objectes
- Controlar i coordinar les activitats d'altres objectes

3.3. Criteris GRASP

- **Principis generals (GRASP)**: descripció dels principis bàsics d'assignació de responsabilitats a les classes expressades com a patrons (són molt generals)

General Responsibility Assignment Software Patterns (GRASP)

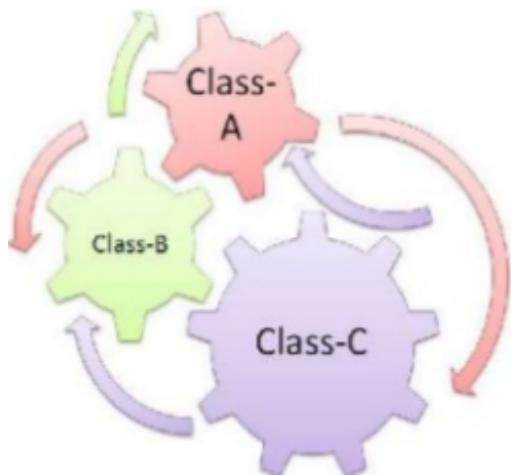
Distribuir responsabilitats és la part més difícil del disseny OO. Consumeix una bona part del temps

- Patrons/Criteris **GRASP**:
 1. Baix Acoblament/Alta Cohesió
 2. Expert en Informació
 3. Creador
 4. Controlador
 5. Fabricació Pura
 6. Indirecció
 7. Polimorfisme
 8. Variacions Protegides

C. Larman

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (second edition) Prentice-Hall, 2002. (pag. 205 i següents)

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

- **Cohesió:**

Classes amb:

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

del grau de relació i de
ració de les diverses
abilitats d'una classe
, associacions, mètodes,



ALTA Cohesió



Pot donar lloc a més indireccions (o més crides)

3.3.2. Expert en Informació

- **Expert en informació :**

Problema: a quina classe assigno una responsabilitat concreta?



Assignar una responsabilitat concreta a la classe que té la informació necessària para a complir-la



El Client vol saber la quantitat total dels productes del seu “Carrito”, qui té la responsabilitat de fer-ho?

3.3.2. Expert en Informació

Problema: *A quina classe li assigno una responsabilitat??*



Solució

- Assignar la responsabilitat a la classe que té la informació necessària per fer aquella responsabilitat
- Cada objecte és responsable de mantenir **la seva pròpia informació** (príncipi d'encapsulament).
- Si té relacions de **composició** amb altres objectes (les seves parts) també serà el responsable de:
 - conèixer la informació d'ells,
 - crear-los (patró creador)
 - delegar-li les seves operacions.

NOTA: No sempre existeix un únic expert, sinó que poden existir **experts parcials** que hauran de col·laborar.

3.3.2. Expert en Informació



Pros:

- Es manté encapsulament → Baix acoblament
- Conducta distribuïda entre classes → Alta cohesió.
- No es creen classes “Deu”.

Cons:

- Hi ha situacions en què no porta a bons dissenys, usualment per problemes d'acoblament i cohesió. Es trenquen principis arquitectònics (p.e., arquitectura a 3 capes)

*Ex: **Interfície d'usuari**: Visualitzar les línies de venda en una finestra*

*Ex: **Persistència**: Emmagatzemar una venda a la base de dades*

3.3.3. Creator

- **Creador :**

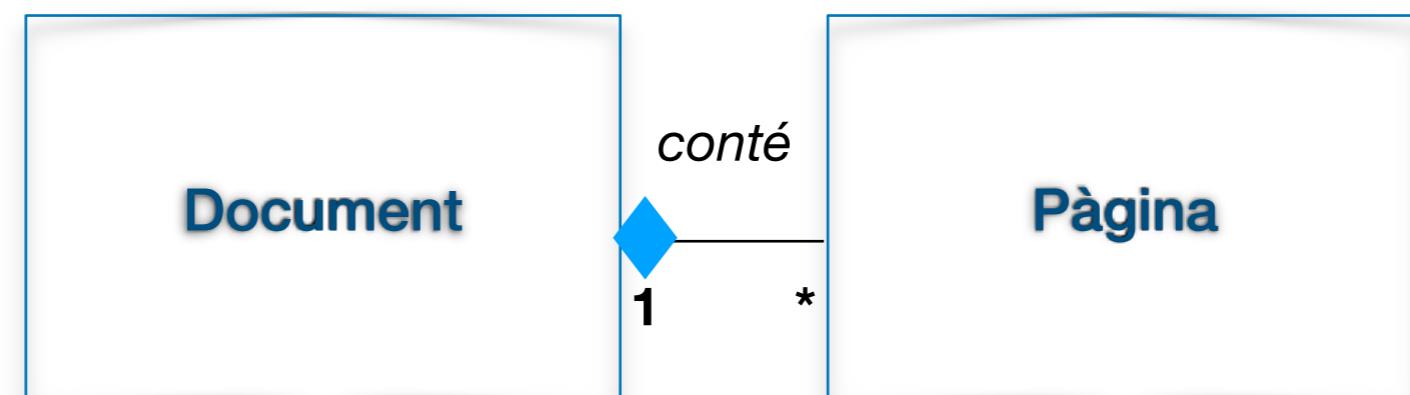
Problema: quina és la classe responsable de crear-ne una altra?



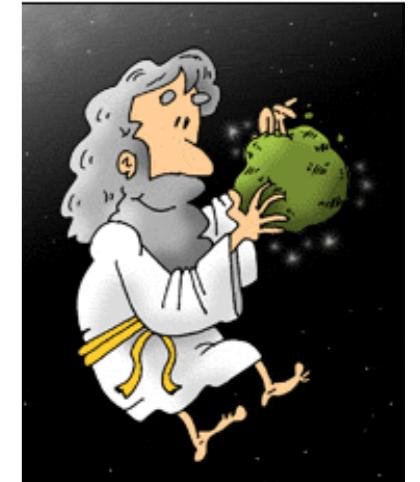
(See Genesis 1:1-2:3)

AND THEN, FOR FUN, GOD CREATED ONE
WITH SUPERHEROES, UNICORNS, LIGERS,
OCEANS OF ORANGE SODA ... AND IT WAS
"COOL" MORE THAN "GOOD"

Assignació de la responsabilitat d'una classe B per a crear la classe A si B necessita està connectat a A en algun event



3.3.3. Creator



Problema:

- quina és la classe responsable de crear-ne una altra?

Solució:

Assignar a una classe **B** la responsabilitat de **crear** una instància d'A si es compleixen una o més de les següents condicions:

- B agrega objectes de classe A
- B conté objectes de classe A
- B manté un registre de les instàncies de la classe A
- B usa molts objectes de classe A
- B conté les dades per inicialitzar A en el moment de la seva creació
- Si més d'una classe compleix les condicions, seleccionar aquella que agrega o conté objectes de classe A

3.3.3. Creator

Pros:

- Es manté encapsulament → Baix acoblament

COPYRIGHT BIBLE GATEWAY



Cons:

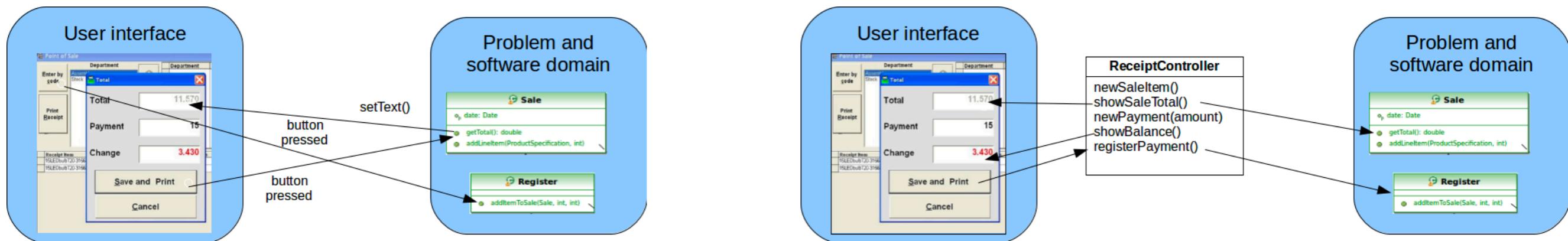
- Sovint la creació d'objectes requereix una complexitat significativa, per motius d'eficiència o d'altres. En aquests casos és raonable delegar la creació a una classe (per això s'utilitza el patró **Factory Template** o **Abstract Factory** (es veurà en la secció 3.5.2)
- Poden portar problemes d'acoblaments alts trencant principis arquitectònics.

3.3.4. Controlador

- **Controlador:**

Problema: Qui és el responsable de manegar els events d'input sobre un objecte que es reben de la interfície o de la capa de presentació?

Assignació de la responsabilitat a un objecte “root” (o controlador) o bé a un “cas d'ús” dins del sistema



3.3.4. Controller

Observacions:

- Un controlador **NO** és un objecte de la interfície
- El Controlador estableix el diàleg i interpretació dels errors del Model per a traduir-los a la Vista.

Pros:

- El Controlador és un Mediador entre la Vista i el Model. Desacobla funcionalitats

Cons:

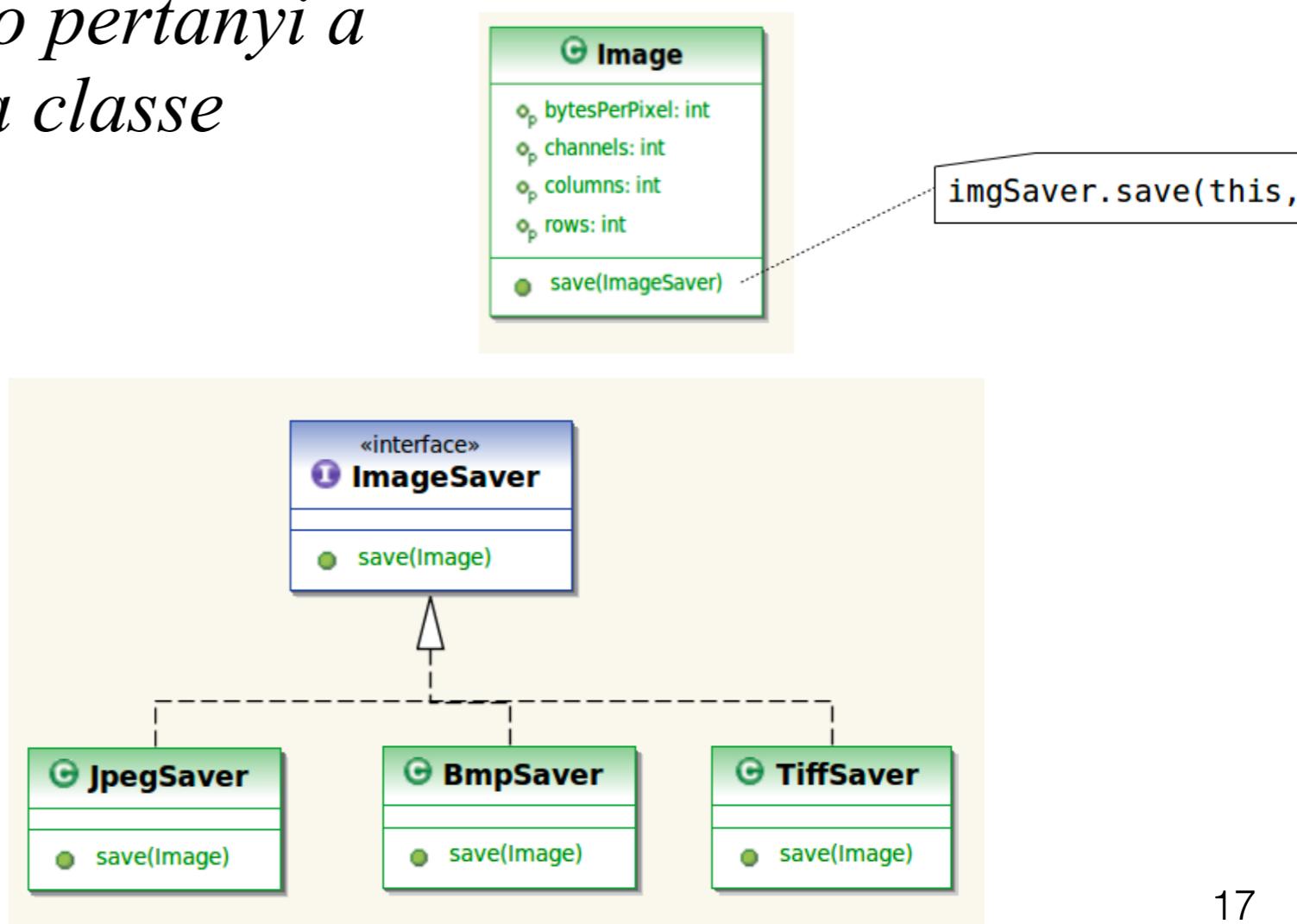
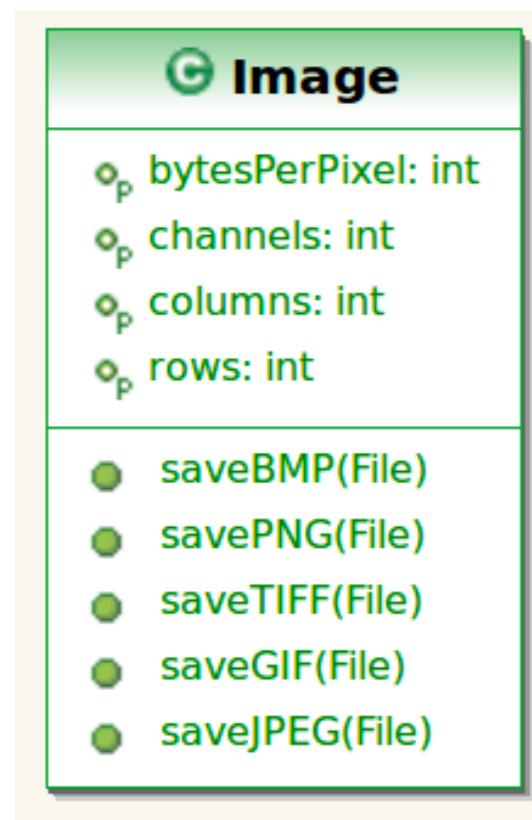
- Baixa Cohesió
- Single Responsibility Principle
- Esdevenir una classe Deu (poca cohesió): Bloated Controllers
 - El Controller pot utilitzar diferents API's (o **façanes**) per a demanar serveis complexes del Model
 - Fer diferents controladors

3.3.5. Fabricació Pura

- **Fabricació Pura :**

Problema: Qui és el responsable quan no hi han classes del Model de Domini que ens aconselli l'expert en informació?

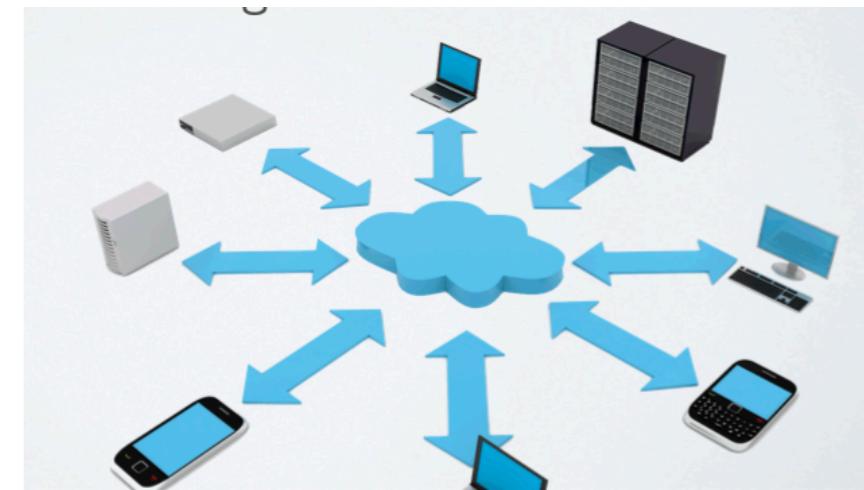
Quan un comportament no pertanyi a ningú altre, crea una nova classe



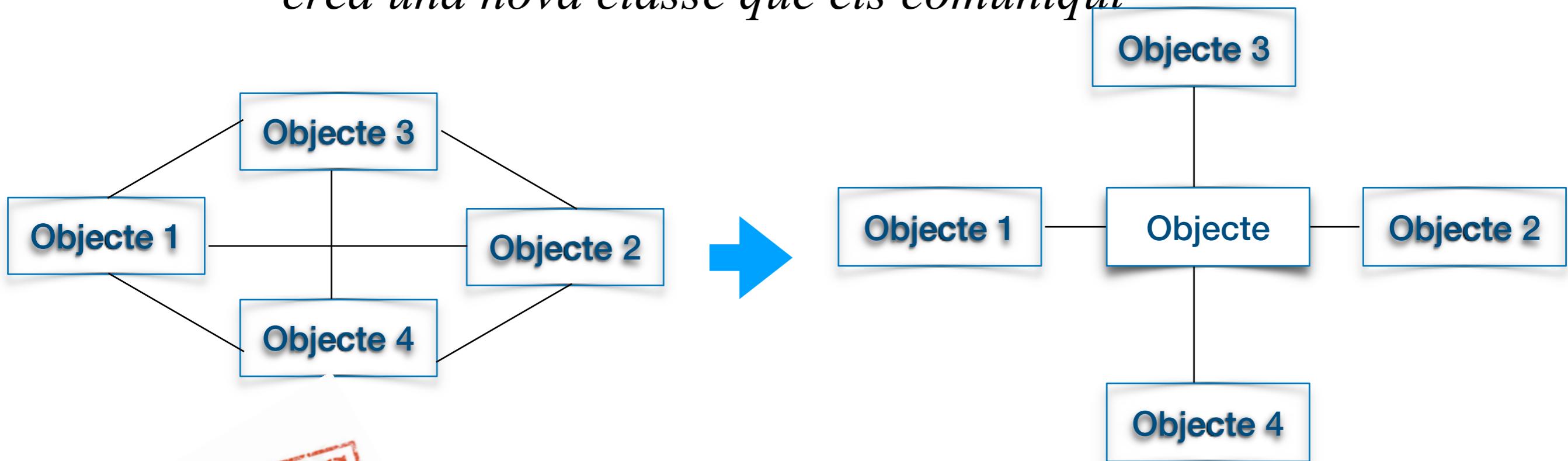
3.3.6. Indirecció

- **Indirecció :**

Problema: Com assignar responsabilitats per evitar acoblament directa entre dues o més classes?



*Per reduir l'acoblament entre objectes,
crea una nova classe que els comuniqi*



WARNING

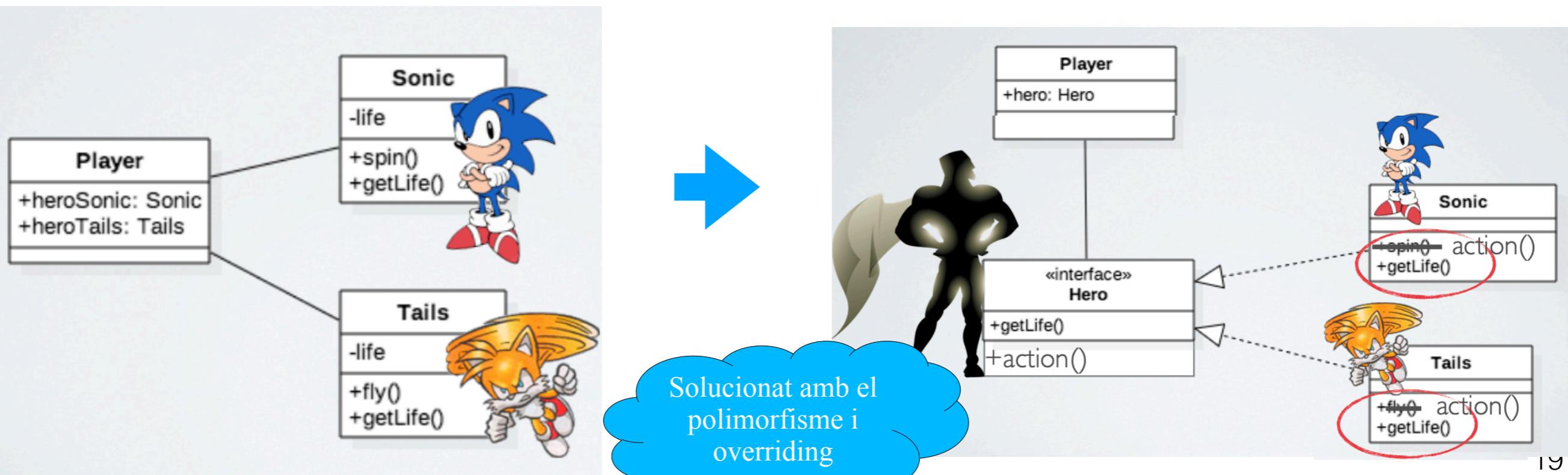
Dóna lloc a més indireccions (o més crides)

3.3.7. Polimorfisme

- **Polimorfisme :**

Problema: com manegar alternatives de comportament basades en el tipus? Com fer components “plugables”?

Permetre noms iguals a mètodes que donen el mateix servei a diferents classes (Subclasses o interfícies). Canvia automàticament el comportament segons el tipus.



3.3.7. Polimorfisme



Observacions:

- Usa interfícies quan vulgis fer polimorfisme sense fer una jerarquia particular de classes
- Aplica el Liskov Substitution Principle (LSP) (veure principis SOLID) per garantir bé les herències, en cas que les usis

Consells:

- Usa polimorfisme en el cas que hagis de fer una part molt variable amb bastanta probabilitat
- Evita usar polimorfisme només “per si un cas” hi han canvis en un futur

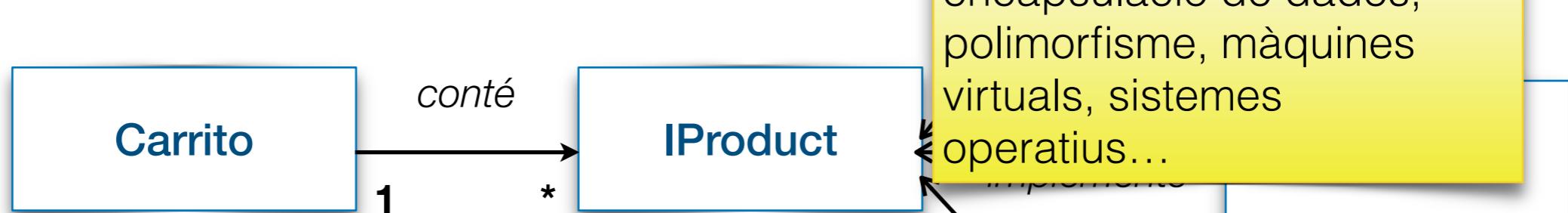
3.3.8. Variacions protegides

- **Variacions protegides :**

Problema: Com dissenyar objectes, subsistemes per a que les variacions o canvis en aquests elements no suposi un impacte no desitjat en d'altres elements?



Identifica els punts inestables i assigna responsabilitats posant interfícies estables al seu voltant



Llei de Demeter: els objectes no parlen mai amb objectes amb qui no estan connectats directament

Alimentació

3.3. Criteris GRASP

- **Principis generals (GRASP)**: descripció dels principis bàsics d'assignació de responsabilitats a les classes expressades com a patrons (són molt generals)

General Responsibility Assignment Software Patterns (GRASP)

Distribuir responsabilitats és la part més difícil del disseny OO. Consumeix una bona part del temps

- Patrons **GRASP**:
 1. Baix Acoblament/Alta Cohesió
 2. Expert en Informació
 3. Creador
 4. Controlador
 5. Fabricació Pura
 6. Indirecció
 7. Polimorfisme
 8. Variacions Protegides



C. Larman

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (second edition) Prentice-Hall, 2002. (pag. 205 i següents)