**Tema 8: Disseny de Software en UML**

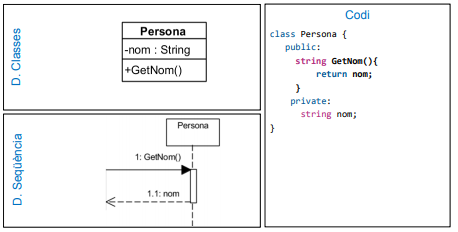
**Elements de Disseny en UML:**

* Crides i resultats.
* Crides entre instàncies.
* Navegabilitat.
* Agregats.
* Polimorfisme.
* Elements Abstractes.
* Creadores.

**Obtenir el diagrama de seqüència de disseny**

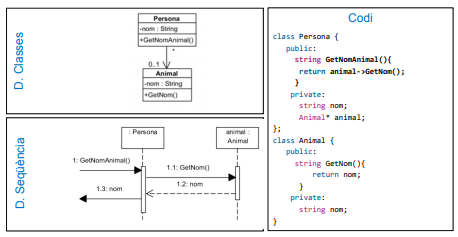
A disseny, el diagrama de seqüència ens ha de mostrar les crides entre les funcions de les diverses classes, per tant, ha de ser un diagrama que entri a fons en les interaccions entre classes. Avís: Alguns elements del nostre diagrama de seqüència, repercutiran en canvis al diagrama de classes.

**Crides i Resultats**

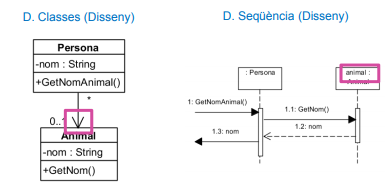


La forma de cridar una funció del diagrama de Seqüència és la mateixa en que es cridaven els esdeveniments a Especificació.

**Crides entre Instàncies**

****

Una instància d’una classe pot cridar funcions públiques de les instàncies amb que està relacionada, privades d’ella mateixa i protegides dels seus ancestres a la jerarquia.

****

Si una classe es comunica amb una altra, ho hem de marcar al diagrama de classes. Això indica la navegabilitat.

Si una classe crida funcions d’una altra amb la que està relacionada, es pot posar el nom de rol/membre per a identificar l’objecte cridat.

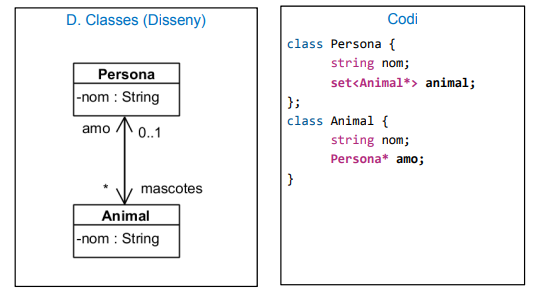
**Navegabilitat**

Indica si és possible o no travessar una associació binària d’una classe a una altra:

* Si A és navegable cap a B, des d’un objecte d’A es poden obtenir els objectes de B amb els que està relacionat.

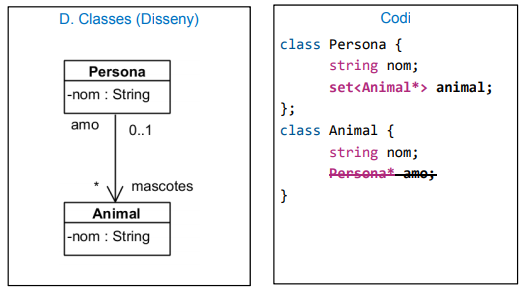
La navegabilitat d’una associació resultant del procés de disseny pot ser bidireccional, unidireccional o no navegable.

Navegabilitat bidireccional:

****

Les relacions navegables de forma bidireccional necessiten mantenir punters/referències a ambdues classes.

Navegabilitat unidireccional:

****

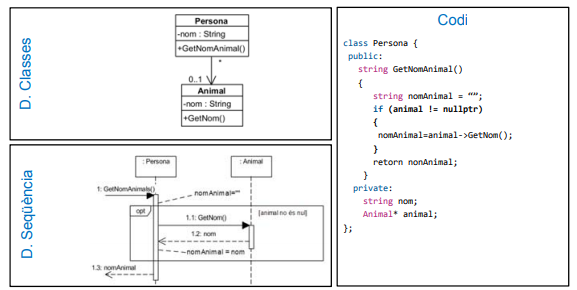
Les relacions navegables de forma unidireccional necessiten mantenir punters/referències a una sola de les classes.

Navegabilitat nul·la:

Si una relació no és navegable en cap sentit, aleshores vol dir que no hi ha cap referència a ella dins del codi.

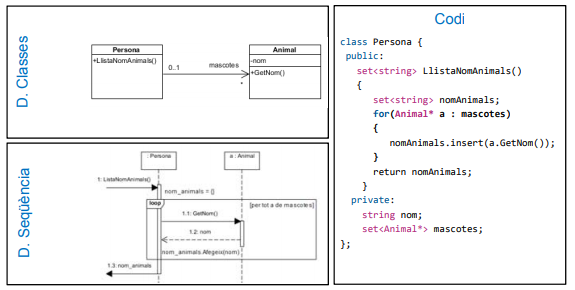
Per tant, la pròpia relació pot desaparèixer del diagrama de classes de Disseny.

**Opcions / Condicionals**



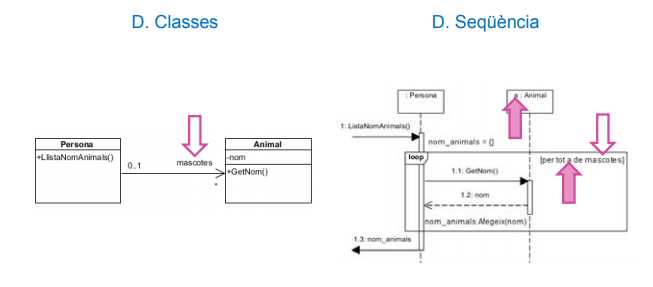
La forma de representar Opcions o Condicions al diagrama de seqüència serà mitjançant l’ús dels frames de opt i alt, de la mateixa manera que es feia als diagrames de seqüència d’especificació.

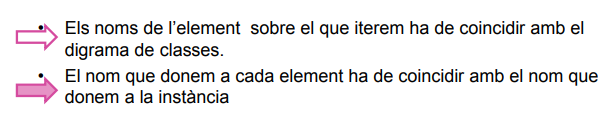
**Agregats**



Un agregat és una col·lecció d’objectes.

La forma de representar iteracions sobre agregats es fa mitjançant al frame de loop de la mateixa manera que es feia als diagrames de seqüència d’especificació.





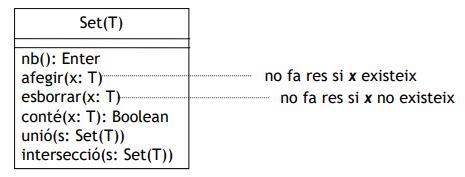
Un agregat és una col·lecció d’objectes.

Sorgeixen en diversos contexts:

* Rols navegables amb multiplicitat més gran que 1.
* Operacions que reben o retornen una col·lecció de valors.
* Atributs multivaluats.

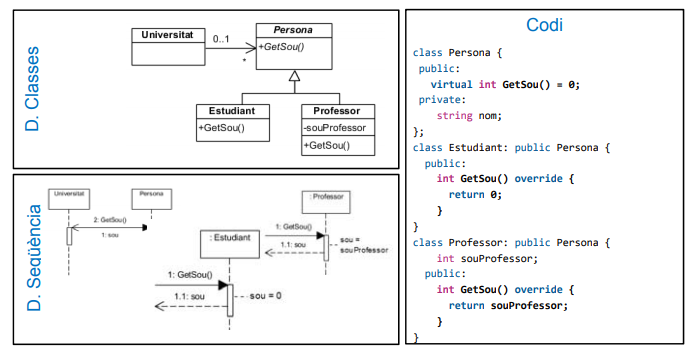
En tots aquests casos, considerem l’agregat com un conjunt (Set):

* S’hi poden aplicar operacions següents (i només aquestes):



**Polimorfisme**

És un mecanisme dels llenguatges de programació que permet que, en el cas de que dues funcions es diguin igual en una jerarquia de classes, aleshores el sistema cridarà en temps d’execució a la funció que es trobi implementada al nivell més inferior a la jerarquia.

****

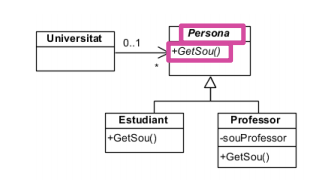
La forma de marcar el polimorfisme en el Diagrama de Seqüència és creant funcions amb el mateix nom dins de la mateixa estructura de Generalització / Especialització.

**Elements abstractes**

Quan una operació dins d’una classe no te cap mena de codi, aquesta s’ha de definir com a abstracta.

Això provoca que:

* Tots els seus descendents dins de la jerarquia estan obligats a tenir aquesta funció.
* La pròpia classe esdevé abstracta i no es poden declarar instàncies que siguin exclusivament del seu tipus.
* Una superclasse només pot ser abstracta si la seva especialització és complete.

****

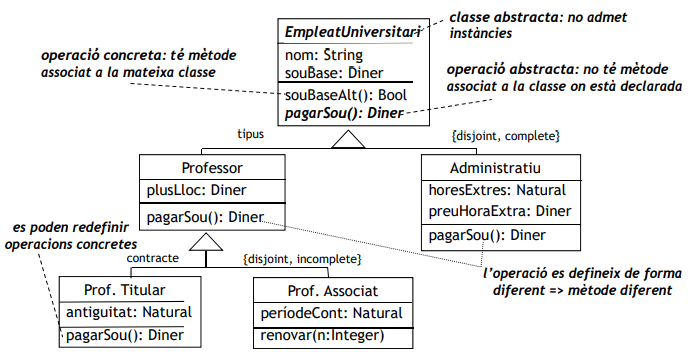
La forma de marcar elements abstractes és posant el nom en cursiva.

A IES podeu posar una nota al costat de la classe/funció per a indicar-ho.

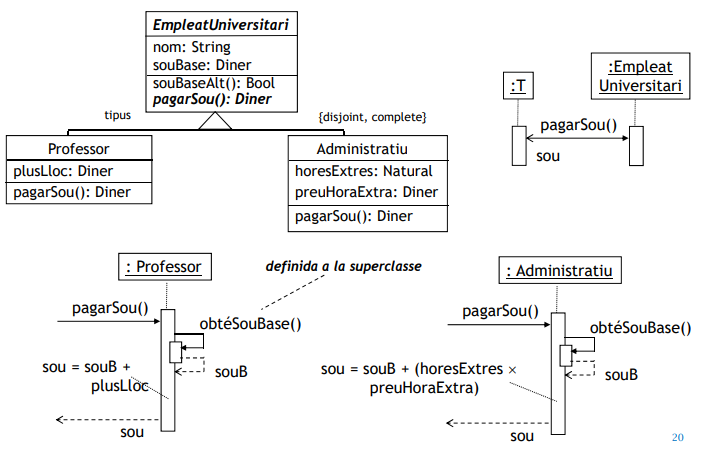
Extra: Aquesta és una manera de tenir especialitzacions / generalitzacions de tipus Complete.

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.

****

Vinculació dinàmica:



**Creadores**

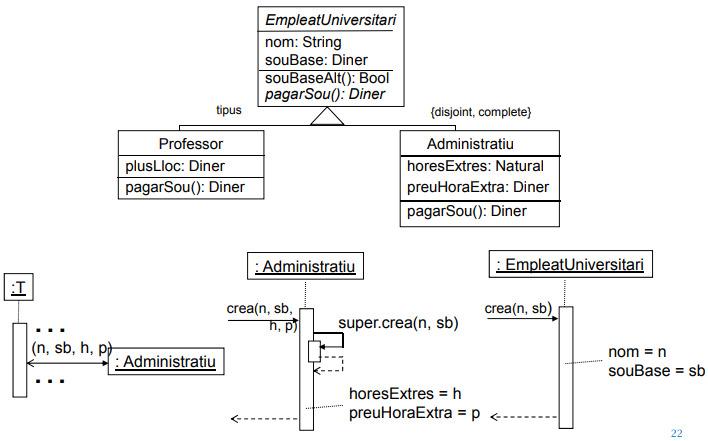
La forma de crear instàncies als llenguatge de programació és, normalment, mitjançant l’ús d’unes funcions estàtiques especials anomenades Creadores.

Una creadora rep com a paràmetres alguns (o tots, o cap) dels atributs de la classe que ha de crear i en retorna una instància.

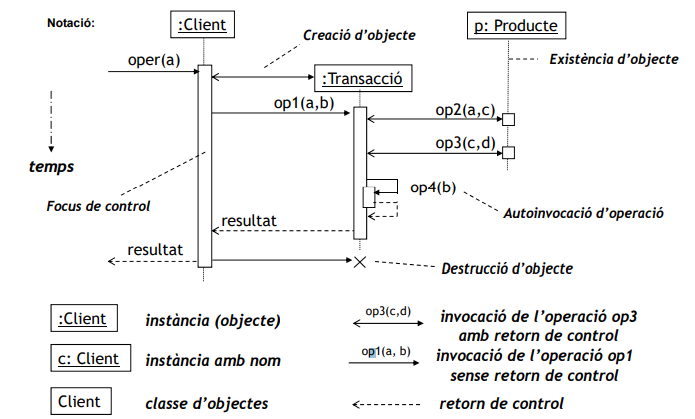
En el cas de les creadores de subclasses d’una jerarquia, estan obligades a cridar a la creadora de la seva superclasse per a completar la seva construcció.

A IES assumirem la existència d’una creadora amb tots els atributs de la classe com a paràmetres.

Creació d’objectes en una jerarquia d’herència:



**Diagrames de seqüència: sintaxi completa**



Noms dels objectes: batejar-los quan calgui per identificar el seu origen:

* si són resultat d’una operació, usar el mateix nom en el resultat i en l’objecte.
* si s’obtenen recorrent una associació amb multiplicitat 1, usar el nom del rol.
* si han arribat com a paràmetres, usar el nom dels paràmetres.

Paràmetres de les operacions:

* cal indicar explícitament amb quins paràmetres s’invoquen les operacions.

Resultats de les operacions:

* donar nom al resultat si surt en algun altre lloc del diagrama.
* també als valors retornats en els paràmetres.

Comentaris:

* no deixar cap aspecte rellevant sense comentar.
* en particular, ha de quedar clar:
  + com es calculen els resultats de les operacions.
  + Quins valors es passen com a paràmetres a les operacions creadores.
  + com es modifiquen els valors dels atributs i pseudo-atributs de les classes.
  + usar noms d’atributs, associacions, etc.
  + usar sintaxi Java per a operacions aritmètiques.

No cal especificar el comportament de les operacions getter i setter.

Convencions:

