UML (I): el model estàtic

Benet Campderrich Falgueras
Recerca Informàtica, SL

P00/05007/00300

UOC
www.uoc.edu

Índex

Introducció

Com sabem, per a l'anàlisi i el disseny orientats a objectes farem servir els conceptes i les notacions –essencialment gràfiques– d'UML.

Vegeu el mòdul "Introducció a l'enginyeria del programari OO" d'aquesta assignatura.

UML comprèn un cert nombre de diagrames interrelacionats mitjançant conceptes comuns; aquests diagrames, només per a descriure'ls, els considerarem agrupats en tres models:

- El **model estàtic**, que descriu l'estructura de classes i objectes.
- El que podríem anomenar **model dinàmic** o de **model de comportament**, que descriu les interaccions entre els objectes dins el programari.
- El **model d'implementació**, que descriu l'estructura del programari en termes dels components de què consta i de llur ubicació.

En aquest mòdul veurem el model estàtic. En mòduls posteriors tractarem el model dinàmic, el model d'implementació i la utilització d'UML en l'anàlisi i en el disseny.

Vegeu el model dinàmic i el model d'implementació al mòdul "UML (II): el model dinàmic i d'implementació" d'aquesta assignatura.

El model estàtic consta, d'una banda, de classes i objectes, i de l'altra, de relacions de diferents menes entre classes i entre objectes.

Objectius

L'objectiu principal d'aquest mòdul és conèixer en profunditat el diagrama estàtic d'UML, cosa que vol dir tant entendre'n els conceptes com saber-los aplicar, i més detalladament implica el següent:

1. Aprofundir en el concepte de classe, en especial quant als diferents tipus de classes i conceptes relacionats, i familiaritzar-se amb les notacions respectives.

2. Entendre els tipus més freqüents de relacions entre classes i saber-los identificar.

3. Conèixer els principals elements complementaris d'UML aplicables al diagrama estàtic.

1. Concepte de model estàtic i diagrama de classes

El model estàtic de l'UML és aquell en què es descriuen les classes i els objectes. S'anomena *estàtic* perquè mostra totes les relacions possibles al llarg del temps, no quines són vàlides en un cert moment.

Exemple de model estàtic

Un diagrama estàtic ens pot mostrar que cada professor té almenys una assignatura i que cada assignatura té almenys un professor, però no ens diu quines assignatures ensenya un professor concret.

El model estàtic consta dels dos diagrames següents:

- El **diagrama de classes**, que pot contenir classes i objectes i relacions entre aquests, i que es fa sempre.

- El **diagrama d'objectes**, que només conté objectes i relacions entre aquests i és opcional, ja que es fa servir principalment per a fer exemples del diagrama de classes amb objectes concrets d'aquestes.

Un diagrama de classes mostra l'estructura estàtica de les classes en un **domini** (porció del món real considerada per una aplicació); s'hi mostren les classes i les relacions entre aquestes, que poden ser d'herència, associació, agregació o ús.

Ús del model estàtic

El model estàtic es fa servir en totes les etapes del cicle de vida; a les diferents etapes es documenten diferents tipus d'objectes. En l'anàlisi es consideren objectes del món de l'usuari (per exemple: articles, factures, clients, etc.) i en el disseny, en canvi, es consideren objectes de la tecnologia informàtica: pantalles, gestors de disc, etc.

El model estàtic pretén ser independent del llenguatge de programació, però, tanmateix, si se sap quin serà, és convenient de no emprar-lo en l'anàlisi de conceptes que sabem que el llenguatge en qüestió no suporta, si volem estalviar-nos de fer molts canvis en arribar al disseny. També caldrà tenir en compte que, quan UML permet de descriure coses que són incompatibles amb un llenguatge determinat, rarament l'eina CASE ens ho impedirà; serà, per tant, responsabilitat del programador evitar de caure en la utilització de conceptes no suportats pel llenguatge de programació.

Relacions en els llenguatges de programació

Els llenguatges de programació suporten, les relacions d'herència però no distingeixen entre els altres tres tipus de relacions que hi ha; per tant, en passar del disseny a la programació aquests tipus s'hauran de transformar.

També pot passar el contrari; és a dir, que es vulguin modelar coses que l'eina CASE no suporta, perquè UML no les preveu, o per altres raons; llavors caldrà documentar aquests aspectes mitjançant comentaris lliures, que totes les eines permeten. Algunes eines permeten que l'usuari defineixi extensions, però si una empresa fa servir aquesta possibilitat els diagrames generats no seran portables a altres empreses.

2. Classificadors

El **classificador** és l'entitat bàsica del model estàtic. Un classificador és quelcom més general que una classe; és un conjunt els elements del qual es diuen **instàncies**.

El classificador com a tal no té símbol gràfic, sinó que tenen els seus estereotips: classe, tipus de dada i interfície.

Estereotip

Un estereotip d'un element d'UML és una variant més restrictiva de l'element en qüestió; hi ha estereotips que formen part d'UML i hi pot haver també estereotips definits per qui fa el diagrama, que són un instrument per a estendre UML però perdent portabilitat.

- El concepte de **classe** és el que ja coneixem de la programació orientada a objectes, i les seves **instàncies** són els objectes, els quals tenen identitat, en el sentit que fins i tot dos objectes que coincideixen en el valor de tots llurs atributs són objectes diferents si s'han creat com a tals.

- Per **tipus de dada** entenem un tipus base ofert per algun llenguatge de programació o construït pel programador; té operacions associades igual que les classes, però les seves instàncies, a diferència dels objectes, no tenen identitat.

- Una **interfície** descriu només aquelles operacions d'una classe que són visibles des d'altres classes; es diu que la classe en qüestió implementa la interfície corresponent.

La utilitat del concepte de classificador rau en el fet que els estereotips esmentats tenen moltes coses en comú, que n'hi ha prou amb indicar una vegada en el classificador. La notació gràfica simplificada és la mateixa per a tots tres: un rectangle.

Vegeu una notació gràfica ampliada de les classes en l'apartat 4 d'aquest mòdul didàctic.

Tots els classificadors han de tenir un nom. En un classificador es pot indicar la paraula clau de l'estereotip (entre cometes llatines, «»); quan no s'indiqui cap estereotip, es tractarà d'una classe.

Exemple de classe i interfície

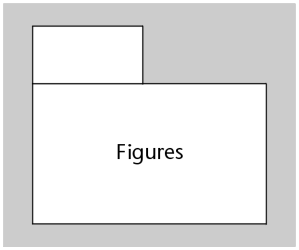
3. Paquets

Un **paquet** o ***package*** és només una "caixa" que conté elements com ara classificadors, objectes o altres paquets, i també altres entitats que veurem més endavant, com ara casos d'ús.

Paquets en Java

Per la definició que donem de paquet podem veure que el concepte de paquet en UML és diferent –i més ampli– que en Java.

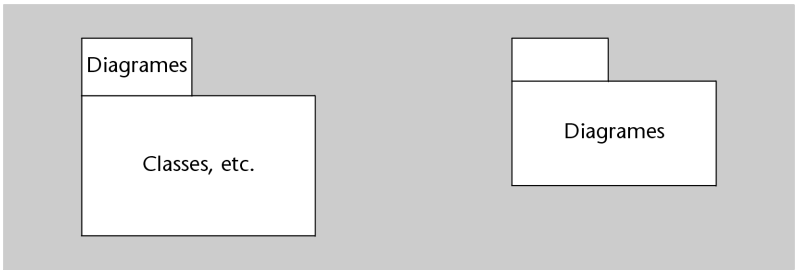
Gràficament un paquet és representa així:



Totes les aplicacions han de tenir almenys un paquet que normalment s'anomena *arrel*. Cada element d'un paquet té visibilitat, és a dir, pot ser reconegut o bé des de tots els altres paquets, o bé només d'alguns.

Exemple de paquets

Aquestes són dues maneres de representar el mateix paquet:

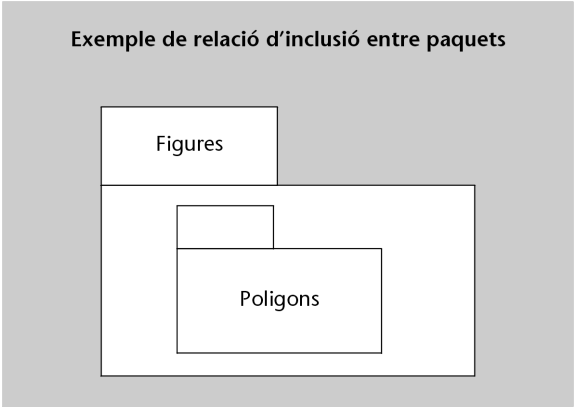


En la primera, es poden incloure dins el símbol del paquet els símbols dels elements que conté; la segona, simplificada, és més adequada per a representar referències al paquet (des d'altres paquets, per exemple).

De **relacions entre paquets**, hi poden haver els tipus següents:

- **D'especialització**: si un paquet *A* hereta d'un altre *B* tots els elements de *B*, són casos més restrictius d'elements d'*A*.

- **D'inclusió**: si el paquet *A* inclou el *B*, tots els elements de *B* estan també a *A*.



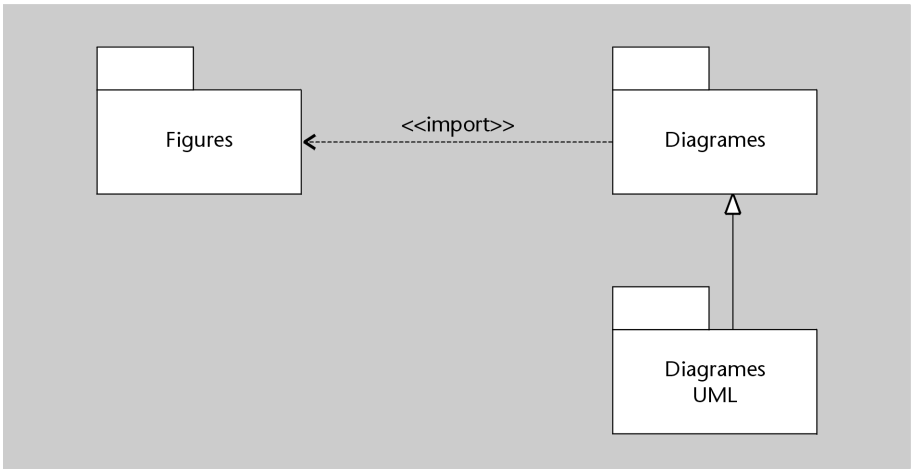
- **D'importació**: des del paquet que importa es reconeixen els noms dels elements de l'altre paquet visibles des de l'exterior*.

* Per exemple, públics en el sentit de Java.

- **D'accés**: no solament es reconeixen els noms dels elements sinó que, a més, es poden fer servir.

Exemple de relacions entre paquets

En la representació gràfica, el paquet *Diagrames* que veiem a continuació importa del paquet *Figures*, i el paquet *Diagrames d'UML* hereta del paquet *Diagrames*; una relació d'accés es representaria amb la paraula clau *access*.



4. Classe i conceptes afins

De la programació orientada a objectes sabem que una **classe** descriu un conjunt d'objectes que tenen tots els mateixos atributs i les mateixes operacions; d'atributs i operacions n'hi ha d'**instància**, és a dir, lligats a objectes individuals, i de **classe**, que no estan relacionats amb cap objecte en particular de la classe. Aquest mateix és el concepte de classe en UML.

En principi, cada classe és visible (és a dir, reconeguda) dins el paquet on s'ha declarat i el seu nom no pot estar repetit en aquest paquet; però des d'un paquet es reconeixen els noms de classes –i elements en general– d'un altre paquet del qual s'importa, en el sentit indicat més amunt; llavors el nom de la classe ha d'anar qualificat pel nom del paquet, així: Paquet::Classe.

4.1. Representació ampliada de les classes

Com que una classe és un classificador, es pot fer servir com a símbol de la classe un simple rectangle amb el nom; ara bé, com que una classe és un encapsulament d'uns atributs i unes operacions, hi ha també una representació gràfica més detallada per mitjà d'un rectangle dividit en els tres compartiments següents:

Recordeu...

... que com a sinònims d'operació es fan servir sovint *mètode* i *servei*; però en UML el terme *servei* no es fa servir i per *mètode* s'entén la implementació d'una operació.

- El primer compartiment conté el nom de la classe.
- El segon compartiment conté la llista dels atributs.
- El tercer compartiment correspon als serveis de la classe.

L'usuari pot crear altres compartiments a més dels tres obligatoris, per a donar informació addicional, com ara excepcions, requisits, etc.

4.1.1. El compartiment del nom

A dalt de tot del compartiment de la classe es pot indicar un estereotip*; hi ha alguns estereotips que formen part d'UML –com ara *metaclass*, del qual es parlarà més endavant–, i se'n poden definir per a un projecte concret, per exemple.

*** Aquí es tracta d'estereotips de la classe; recordem que la classe, per la seva banda, és un estereotip del classificador.**

Just a sota hi ha el nom de la classe. Es recomana que el nom de la classe sigui un substantiu en singular que de vegades pot tenir una segona paraula que la qualifiqui; es recomana que comenci amb majúscula.

Vegeu les metaclasses al subapartat 4.3.3 d'aquest mòdul didàctic.

El nom de les classes

Els noms de les classes han d'estar ben pensats, ja que la possibilitat de reutilització en depèn molt, perquè quan vulguem reutilitzar una classe d'una llibreria que en tingui centenars l'única pista que en tindrem és el nom, i si no se li va donar un nom adequat serà molt difícil de trobar.

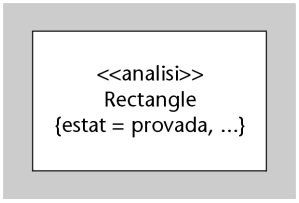
A sota del nom hi pot haver comentaris optatius entre claus ({}) anomenats *cadenes de caràcters de propietats* (*property strings*) o *valors etiquetats* (*tagged values*); els punts suspensius que hi pot haver al final d'un dels apartats indiquen que hi ha més elements però que no es veuen.

Format dels comentaris

Property strings i *tagged values* tenen la forma nom = *valor*; en el cas de propietats booleanes, el nom sol indica la presència de la propietat.

Exemple de classe només amb el compartiment del nom

A la figura següent l'estereotip *anàlisi* podria indicar que la classe *Rectangle* ha estat identificada en l'etapa d'anàlisi.



4.1.2. Especificació dels atributs

Cada atribut té un nom o identificador i un tipus; el tipus pot ser un tipus simple del llenguatge de programació (si més no durant el disseny i la programació, perquè durant l'anàlisi pot ser que les classes es descriguin sense pensar en cap llenguatge de programació en concret), com ara enter o caràcter, o bé un tipus complex, com ara una llista d'enters, o també una classe ja definida.

Un atribut, sigui d'instància o de classe, es defineix així:

visibilitat nom ':' expressió-de-tipus '=' valor-inicial '{' property string '}'

S'indica que un atribut és **atribut de classe** subratllant-ne la definició.

Atribut de classe

Un atribut de classe és el mateix que un atribut *static* de Java i una variable de classe d'Smalltalk i C++.

La **visibilitat d'un atribut** indica fins a quin punt les operacions d'altres classes poden accedir a l'atribut, i s'indica amb els símbols següents:

- public: "+"
- protegit: "#"
- privat: "-"

També tenen visibilitat altres elements del model estàtic, com les operacions i els extrems d'associació.

En el lloc dels atributs es poden fer servir *property strings* que són, respectivament *public*, *protected* o *private*. Les *property strings* són opcionals; a més de les esmentades hi pot haver *frozen*, que indica que no es pot canviar el valor de l'atribut.

La visibilitat

En UML no hi ha definicions del significat d'aquestes opcions de visibilitat, sinó que es deixa per als llenguatges de programació; si algun llenguatge inclou més tipus de visibilitat, també es podran indicar aquí.

Pel que fa al nom dels atributs cal tenir en compte les pautes següents:

- Es recomana que comenci amb minúscula.
- Quan es tracti d'un atribut derivat (això és, que és redundant amb altres, a partir dels quals se'n pot obtenir el valor) el nom ha d'anar precedit de "/".
- És convenient que el nom compleixi les regles lèxiques del llenguatge de programació, si no volem que s'hagi de canviar en arribar al disseny; l'expressió de tipus i el valor inicial també les hauran de respectar.

Es poden fer servir **indicadors de multiplicitat** com en el cas dels vectors o matrius d'acord amb el llenguatge.

Ús dels indicadors de multiplicitat

Considerem:

fills [0..3]: persona

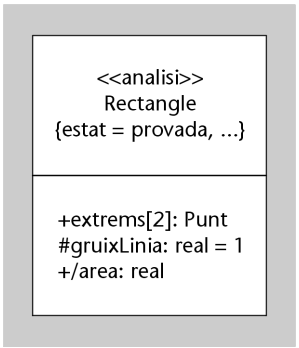
o bé,

fills [3]: persona;

en el primer cas hi podria haver entre 0 i 3 fills, i en el segon n'hi ha d'haver exactament tres.

Exemple de classe amb compartiment de nom i compartiment d'atributs

Com es pot veure a la figura següent:



extrems serien les coordenades de dos vèrtexs oposats del rectangle, que el determinen; *Punt* seria una classe descrita en el mateix paquet; *gruixLinia* té el valor 1 per omissió, i *area* és un atribut derivat, ja que el seu valor es pot calcular a partir de les coordenades dels punts d'*extrems*.

4.1.3. Especificació de les operacions

Una **operació** es defineix així:

Operacions de classe

Una operació de classe equival a una operació *static* de Java, un mètode de classe d'Smalltalk i una *static member function* de C++.

visibilitat nom '(' llista-de-paràmetres '):' tipus-de-retorn '{'property string'}'

S'indica que una operació és **operació de classe** subratllant-ne la definició. La visibilitat s'indica com en el cas dels atributs.

Convé que el nom de l'operació i dels paràmetres i el tipus dels paràmetres i del retorn compleixin les regles del llenguatge de programació. Es recomana que els noms de les operacions comencin amb minúscula.

En la programació...

... convé que els noms de les operacions estiguin ben pensats, perquè són la base del polimorfisme, en el qual s'aplica que "a igual concepte, igual nom".

La **llista de paràmetres** es compon de paràmetres separats per comes; la sintaxi de cadascun és la següent:

mena nom ':' expressió-de-tipus '=' valor-per-omissió

on *mena* és *in*, *out* o *inout* (per omissió, *in*), *nom* és el nom del paràmetre formal; *expressió-de-tipus* és depenent del llenguatge; *valor-per-omissió* és depenent del llenguatge i opcional.

El *tipus-de-retorn* només s'ha de fer servir quan l'operació torni un valor com a resultat, i també es pot fer servir un paràmetre *out* en el seu lloc.

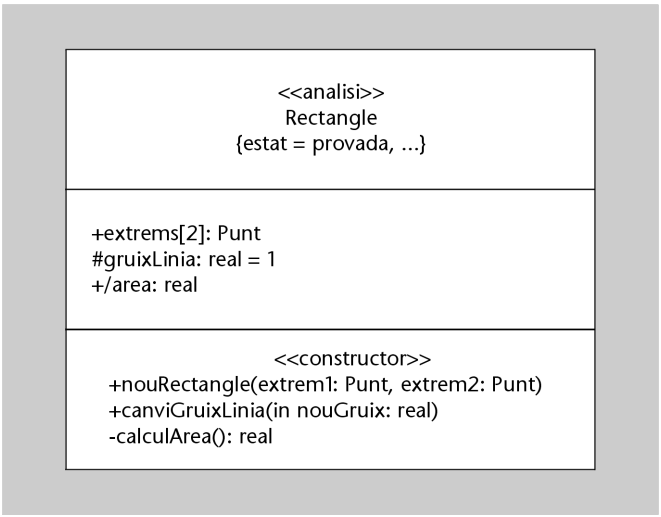
Opcionalment hi pot haver *property strings*: *query* denota que l'operació no modifica l'estat del sistema, i per a especificar la semàntica de concurrència es poden fer servir una d'aquestes: *sequential*, *guarded** o *concurrent*.

*** *Guarded* indica que l'execució pot ser o bé concurrent o bé seqüencial segons es compleixi o no una condició.**

Es poden emprar estereotips posant-los damunt l'operació afectada.

Exemple de classe amb els tres compartiments

Com es pot veure en el gràfic següent, l'operació *nouRectangle* és de l'estereotip (no inclòs dins UML) *constructor* i té dos paràmetres *in* (per omissió); l'operació *calculArea* no té paràmetres però retorna un valor, del qual s'indica el tipus.



4.2. L'herència en l'anàlisi i el disseny

Sabem que l'herència pressuposa que hi hagi dues classes, de les quals una fa el paper de superclasse i l'altra el de subclasse.

La subclasse comprèn un subconjunt dels objectes de la superclasse, els quals, per tant, tenen tots els atributs i operacions d'instància de la superclasse (es diu que la subclasse els *hereta*) i, a més, en poden tenir alguns d'addicionals, específics de la subclasse.

La relació entre una subclasse i la seva superclasse es diu que és una relació *is_a_kind_of*.

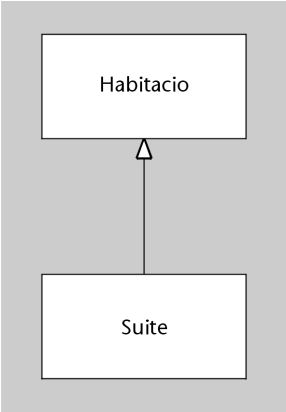
Segons es defineixi primer la superclasse o les seves subclasses, tenim respectivament dos tipus d'herència: herència per especialització i herència per generalització.

4.2.1. Herència per especialització

Es diu d'aquesta manera perquè el que es fa és crear una classe més especialitzada (això és, més restrictiva) a partir d'una classe definida abans.

Exemple d'especialització

Considerem que en la gestió d'un hotel identifiquem la classe *Habitacio* i després ens adonem que hi ha una categoria especial d'habitacions que té atributs i/o operacions diferents, que són les *suites*; això es representa així:



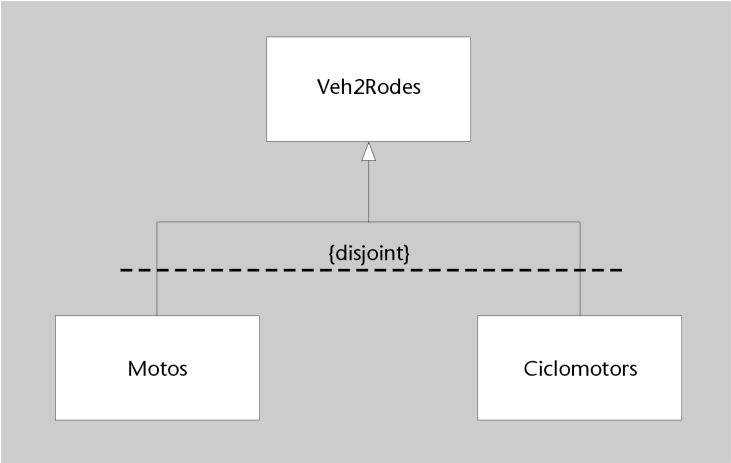
La fletxa amb punta triangular buida i cos continu expressa una relació entre subclasse i superclasse, i el seu sentit indica quina és cadascuna. També es diu que hem creat una **jerarquia d'herència**, molt senzilla en alguns casos* però que esdevé un arbre de diversos nivells si hi ha superclasses que tinguin diverses subclasses i subclasses que siguin alhora superclasses d'altres; fins i tot la jerarquia esdevé una xarxa si hi ha classes que tinguin més d'una superclasse (**herència múltiple**).

*** Com en el cas de l'exemple d'especialització.**

El procés pel qual reconeixem una subclasse dins una altra classe –que conseqüentment passa a ser superclasse de la primera– se li diu **especialització** o **derivació**.

4.2.2. Herència per generalització. Classes abstractes

Suposem que en informatitzar els impostos d'un municipi hi ha, d'una banda, els impostos sobre les motos, i de l'altra, els impostos sobre els ciclomotors; per bé que inicialment s'havia considerat que *Motos* i *Ciclomotors* són dues classes diferents, després s'ha observat que tenen alguns atributs i operacions en comú, i en conseqüència es defineix una superclasse comuna *Veh2Rodes*:



En aquest cas hem procedit a l'inrevés: a partir de les subclasses hem trobat la superclasse; a aquest procés se li diu **generalització**.

Un altre procediment

En el procés de generalització també es podria haver traçat una fletxa independent des de cada subclasse a la superclasse.

La propietat *disjoint* denota que tot objecte de la superclasse pot pertànyer només a una de les subclasses com a màxim.

De la mateixa manera que en l'"Exemple d'especialització" hi podia haver habitacions que no fossin *suites* i, per tant, hi havia objectes que, de les dues classes, només pertanyien a *Habitacio*, en aquest darrer tot vehicle de dues rodes és o bé una moto o bé un ciclomotor, o sigui, tots els elements de *Veh2Rodes* pertanyen a una o altra de les seves subclasses i, per tant, *Veh2Rodes* és una classe artificial, un simple recurs per a no descriure dues vegades allò que tenen en comú *Motos* i *Ciclomotors*; es diu que *Veh2Rodes* és una classe abstracta.

Una **classe abstracta** és una superclasse de la qual no es poden crear (instanciar) directament objectes, sinó que s'han de crear necessàriament en alguna de les seves subclasses.

Per aquesta raó es diu que les classes abstractes són no instanciables.

Una classe abstracta pot tenir **operacions abstractes**, que són aquelles que només estan implementades a les subclasses, en principi de manera diferent a cadascuna d'aquestes.

4.3. Variants en el concepte de classe

En aquest subapartat considerarem diversos tipus especials de classes, els quals no tots es poden representar directament en UML.

4.3.1. Classes diferides

Les classes diferides són classes abstractes que tenen alguna operació abstracta.

També s'anomenen *classes virtuals* perquè en alguns llenguatges com C++ o Delphi els serveis de les característiques que acabem d'indicar es declaren *virtual*; en Java es declaren *abstract*, i en Eiffel *deferred*. En UML es defineixen simplement com a *classes abstractes*.

4.3.2. Classes terminals

Moltes vegades, i especialment en *frameworks*, interessa de bloquejar els canvis que es podrien fer amb l'herència, perquè en crear una subclasse és fàcil que no es coneguin totes les dependències i restriccions que s'hereten, i es poden cometre errors. Java permet de qualificar diversos elements com a terminals (*final*):

- **Classes terminals**, que són aquelles que no poden tenir subclasses.

- **Mètodes terminals**, que són aquells que no poden ser modificats en una subclasse.

- **Atributs terminals**, als quals no se'ls pot canviar la visibilitat en una subclasse.

UML no té aquests conceptes.

4.3.3. Metaclasses

Les metaclasses són classes, cada instància de les quals és una classe.

El concepte de *metaclasse* rarament es troba implementat en els llenguatges orientats a objectes; un llenguatge que l'implementa és el CLOS*, que ha estat creat per investigadors d'intel·ligència artificial i és un derivat del LISP. En UML la metaclasse és un estereotip de la classe.

*** CLOS és la sigla de *Common Lisp Object System*.**

4.3.4. "Classes" parametritzades o plantilles

Una "classe" parametritzada o plantilla* és un descriptor de classe formalment igual a una classe llevat que algun terme de la seva definició sigui un paràmetre.

* En anglès, *template*.

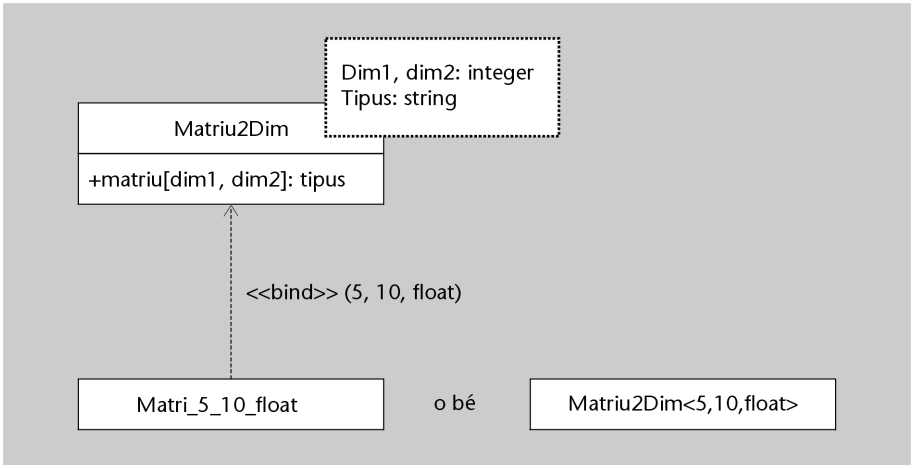
Què és un paràmetre d'una plantilla?

Un paràmetre pot ser, per exemple, el nom d'una operació o atribut, el tipus d'un atribut o d'un paràmetre o del resultat d'una operació, el nombre d'elements d'un atribut que sigui una matriu o el nombre de bytes d'un atribut de tipus *string*.

Quan es donen valors a tots els paràmetres d'una plantilla s'obté una classe totalment especificada, que, per tant, no pot ser modificada directament, però se li poden definir subclasses. Per bé que una plantilla no és pròpiament una classe, es pot definir com a subclasse d'una classe *A*, i aleshores totes les classes que s'hagin obtingut en donar valors als seus paràmetres seran subclasses d'*A*.

Exemple de classe parametritzada

En el diagrama que veiem a continuació hem definit una plantilla, hem generat una classe i hem donat un valor a cada paràmetre; la relació entre la classe i la plantilla es pot representar de les dues maneres indicades:



Les classes genèriques en altres llenguatges

C++ i Eiffel suporten plenament les classes genèriques però amb algunes diferències, mentre que en Java cal simular-les amb la classe *Object*.

Les classes parametritzades s'anomenen *classes genèriques* en alguns llenguatges de programació.

4.3.5. Classes d'utilitat

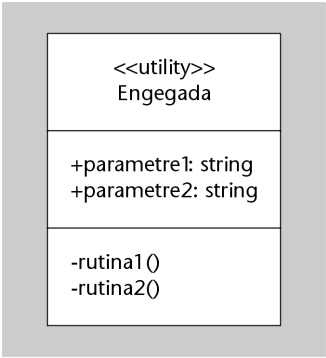
De vegades ens trobem amb rutines que no corresponen a cap operació de cap classe, o bé amb dades que no corresponen a cap objecte determinat, per exemple paràmetres del sistema que només poden tenir un valor cadascun.

Per a incloure aquestes rutines i dades dins un diagrama estàtic d'UML podem definir una classe amb l'estereotip *utility* i incloure-hi les rutines com a operacions i les dades com a atributs.

Com que aquesta classe no tindrà objectes, encara que els atributs i operacions es defineixin formalment com a atributs i operacions d'instància, seran com si fossin de classe i, per tant, no s'hi poden definir ni operacions ni atributs formalment de classe.

Exemple de classe d'utilitat

La classe d'utilitat reuneix diverses coses independents relatives a l'engegada d'un programa: dos paràmetres i dues rutines d'inicialització:



4.4. Interfícies

Una interfície descriu un conjunt d'operacions visibles d'una classe, sense indicar-ne la implementació. Es diu que la classe en qüestió implementa la interfície. Una interfície no és una classe, però equival a una classe abstracta sense atributs i amb totes les operacions diferides.

Recordeu...

... que la interfície i la classe són dos estereotips del classificador.

Les interfícies poden tenir relacions d'herència entre si, però no poden participar en associacions ni tenir estats.

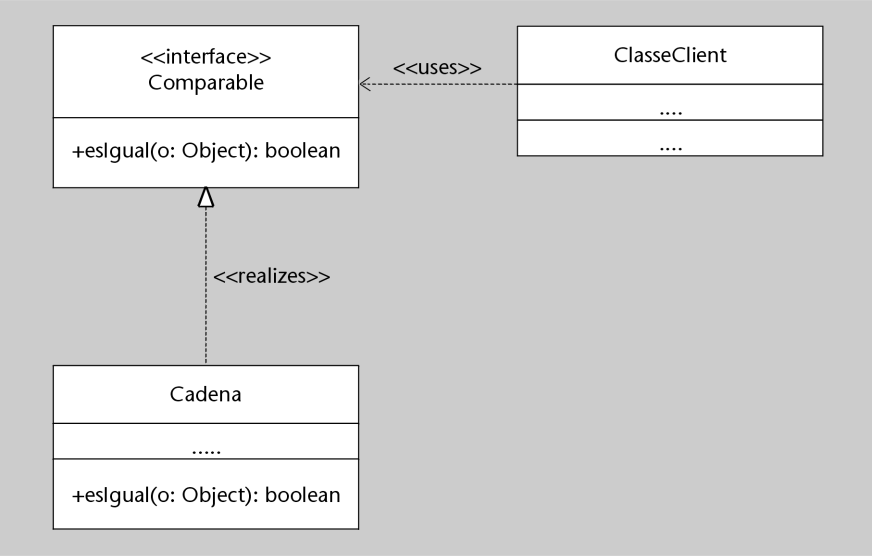
Vegeu les associacions al subapartat 6.1 d'aquest mòdul i els estats en el mòdul "UML (II): el model dinàmic i d'implementació" d'aquesta assignatura.

Cada interfície acostuma a especificar només una part del comportament d'una classe. Una classe pot implementar diverses interfícies, si almenys una d'aquestes no té totes les operacions visibles de la classe; en canvi una interfície només és implementada per una classe.

La notació de la interfície és com la de la classe però amb l'estereotip *interface* i sense el compartiment d'atributs, perquè no en té.

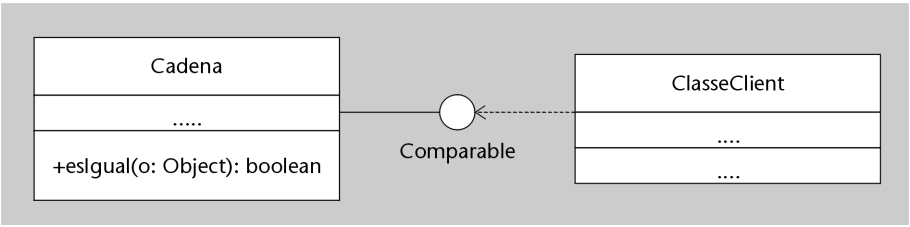
Exemple d'una interfície amb una classe que la fa servir i una altra que la implementa

L'exemple següent s'ha inspirat en la llibreria d'Eiffel:



La interfície *Comparable* pretén declarar totes les operacions que hauran de tenir aquelles classes que vulguin permetre que dos dels seus objectes es puguin comparar segons algun criteri d'igualtat que només queda definit dins la implementació de l'operació *esIgual* dins la classe *Cadena*, que és la que implementa *Comparable*. La classe *ClasseClient* utilitza la interfície *Comparable*, en el sentit que la implementació d'alguna operació de *ClasseClient* crida alguna operació de *Comparable*. Fixeu-vos en com és cada fletxa!

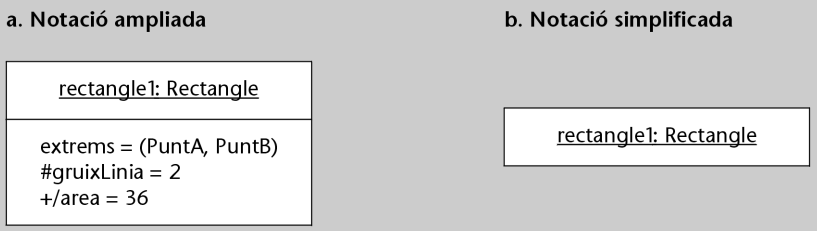
Una notació equivalent més abreujada del mateix exemple és aquesta:



5. Representació dels objectes

Un objecte es representa gràficament d'una manera molt semblant a les classes; s'indiquen els valors als atributs d'instància i opcionalment un nom a l'objecte, que va seguit de ":" i del nom de la classe, tot subratllat. Es pot ometre el tipus dels atributs, i també el compartiment de les operacions, perquè totes dues coses ja es coneixen per l'especificació de la classe.

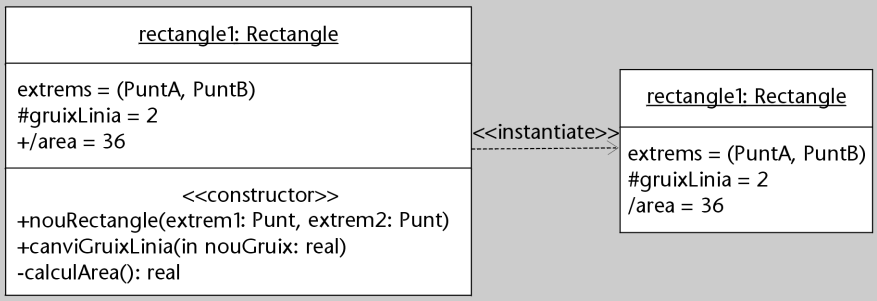
Exemple d'objecte en notació ampliada i en notació simplificada



També es pot representar la relació, anomenada **instanciació**, entre un objecte i la classe a la qual pertany.

Es diu que entre un objecte i la seva classe hi ha una relació *is_a*.

Exemple d'instanciació



Arribats en aquest punt convé que feu els exercicis d'autoavaluació de l'1 al 12 d'aquest mòdul didàctic.

6. Relacions entre classes

Sintàcticament els llenguatges de programació permeten només dos tipus de relacions entre classes: d'herència, que ja hem vist, i client-servidor.

Vegeu les relacions d'herència entre classes al subapartat 4.2 d'aquest mòdul didàctic.

Per **relació client-servidor** s'entén que un objecte (el client) demani a un altre (el servidor), mitjançant un missatge, que executi una operació de les definides a la classe del servidor.

Quant a l'anàlisi i disseny, es consideren d'altres tipus de relacions, perquè les relacions client-servidor són massa simples per a descriure el món real.

Desacord entre autors i l'estàndard de tipus de relació

Lamentablement, no hi ha cap acord entre els autors més importants sobre quins són aquests tipus (i menys encara sobre llur semàntica i notació); ara bé, si UML, esdevingut estàndard, s'acaba imposant, és probable que la seva versió esdevingui acceptada per tothom o gairebé tothom. Però si bé UML permet de modelar gràficament diferents tipus de relacions entre classes (agregació, associació i ús), el significat de cadascun no queda concretat en els documents oficials de l'OMG i, per tant, la decisió sobre quan cal aplicar un tipus de relació o un altre serà sovint una interpretació personal.

6.1. Associacions

Dins d'aquest subapartat veurem els conceptes i terminologia emprats per a designar les associacions i els diferents tipus que hi ha.

6.1.1. Concepte i terminologia

Hi ha una **associació entre classes** quan una classe necessita una o unes altres per a la implementació de les seves operacions, cosa que s'acompleix per mitjà del pas de missatges entre aquestes.

Una associació es defineix partint de la classe, i es concreta en l'existència d'**enllaços*** entre objectes concrets de les classes relacionades per l'associació. Dins una associació es considera que cada classe fa un **paper**** determinat; cada paper té associada una **cardinalitat**. Entre les mateixes classes hi pot haver associacions diferents amb significat diferent.

*** En anglès, *links*.**
**** En anglès, *role*.**

La cardinalitat es defineix al subapartat 6.1.2 d'aquest mòdul didàctic.

Una associació pot tenir nom, que serveix per a identificar-ne el significat, i també es pot donar un nom a cadascun dels papers de les classes.

6.1.2. Associacions binàries i *n*-àries

Associacions binàries són aquelles que tenen lloc entre dues classes. Les dues classes poden ser la mateixa (**associació reflexiva**), i en aquest cas es pot permetre que un objecte estigui enllaçat amb si mateix o que no ho estigui.

En una associació binària, la **cardinalitat** d'un paper *A* és el nombre d'objectes de l'altre paper *B* a què pot estar enllaçat cada objecte d'*A*; s'indica el valor màxim i mínim d'aquest nombre.

Exemples d'associacions binària i reflexiva

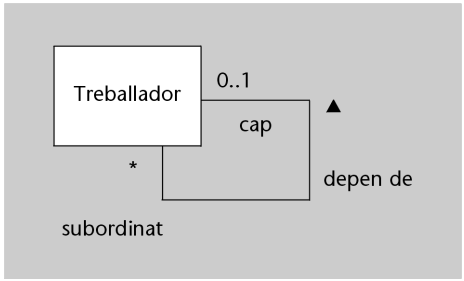
1) Associació binària

Com es pot veure a la associació binària següent:

L'associació significa que una persona treballa en una empresa (no viceversa, observeu el sentit indicat per la punta de fletxa plena); l'empresa fa el paper d'empleador i la persona el d'empleat. Cada persona concreta pot tenir un empleador o cap, mentre que una empresa pot tenir un empleat com a mínim i qualsevol nombre com a màxim, segons indiquen les cardinalitats. La punta de fletxa oberta damunt de la línia de l'associació indica que es pot accedir (navegar) d'una empresa cap als seus empleats.

2) Associació reflexiva

Considerem la figura següent:



El significat d'aquesta associació és que un treballador depèn d'un cap; tant el cap com el subordinat són treballadors. Cada treballador pot tenir com a màxim un cap, mentre que un treballador pot tenir qualsevol nombre de subordinats (l'asterisc tot sol indica que el nombre pot ser qualsevol, inclòs el zero). Un treballador no pot ser cap d'ell mateix, però això no ho indica la notació gràfica.

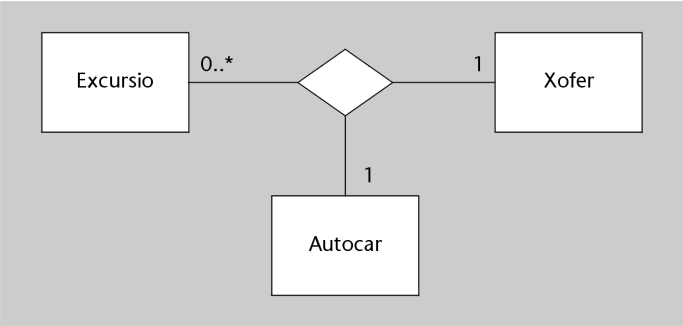
Una **relació ternària** és aquella que té tres papers*, i en general una relació *n*-ària és aquella que en té *n*. Les relacions no binàries, com que no es poden representar per una línia, es representen per un rombe.

*** Això vol dir que hi participen tres classes, diferents o no.**

El significat de la cardinalitat en una associació ternària és el següent: la cardinalitat del paper *A* expressa els límits al nombre d'objectes d'*A* que poden estar enllaçats a cada combinació concreta d'un objecte del paper *B* i un del paper *C*.

Exemple d'associació ternària

Observem l'associació següent:



Un xofer determinat pot conduir un autocar determinat en qualsevol nombre d'excursions ("0..*" és equivalent a "*"), però en una excursió concreta un xofer només pot conduir un autocar, i en una excursió en particular un autocar només té un xofer.

6.1.3. Classes associatives

En principi una associació no és una classe: no té per què tenir atributs ni operacions, i pot no tenir ni nom. Però si una associació ha de tenir atributs i/o operacions propis llavors s'ha de definir com a classe. En aquest cas es parla de *classe associativa*.

Establiment d'enllaços segons una associació

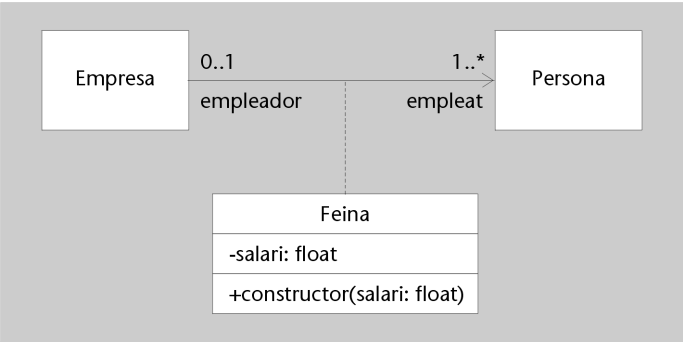
L'establiment d'enllaços entre objectes segons una associació pot ser una operació d'alguna de les classes associades, igual que la navegació d'un objecte a un altre objecte enllaçat a aquest.

Una classe associativa es representa com una classe penjada del símbol de l'associació (la línia, en el cas d'una associació binària, o el rombe en els altres casos) per mitjà d'una línia discontínua.

Exemple de classe associativa binària

El cas que presentem aquí és el mateix que el de l'exemple d'associació binària, llevat que aquí l'associació té un atribut. Com que l'associació ara és una classe, se li ha afegit una operació per a crear instàncies, i també se li ha canviat el nom per un de més adient per a una classe.

Vegeu l'"Exemple d'associació binària" al subapartat 6.1.2 d'aquest mòdul didàctic.

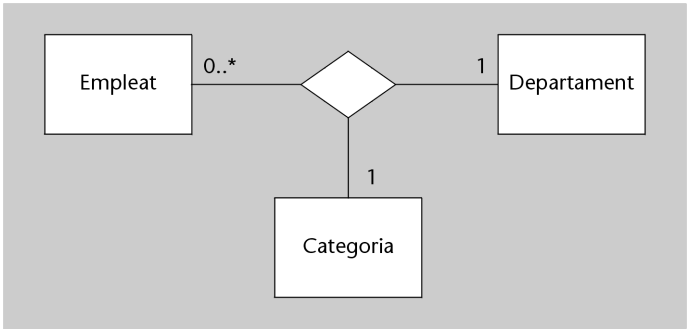


6.1.4. Associacions qualificades

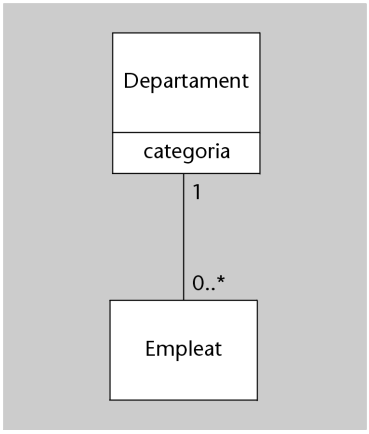
Veurem millor aquest concepte si l'introduïm mitjançant un exemple.

Considerem una classe *Departament* i una altra *Empleats*, que tenen un atribut *categoria*, i volem establir una associació separada entre el departament i els seus empleats per a la categoria *A*, la categoria *B*, etc.; una manera de fer-ho és amb una relació ternària així:

Exemple d'associació binària qualificada i de l'associació ternària equialent.



Aquesta solució és una mica forçada, perquè obliga a convertir un atribut en una classe independent; això es pot evitar mitjançant una associació qualificada en què l'atribut qualificador sigui la categoria de l'empleat. Això es representa així:



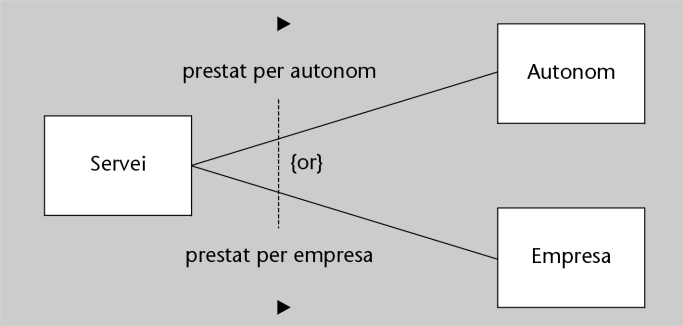
Aquest diagrama es pot interpretar d'aquesta manera: cada empleat té un departament, i d'un departament i categoria determinats hi pot haver qualsevol nombre d'empleats (mentre que si tinguéssim una associació binària ordinària, prescindint de la categoria, qualsevol departament tindria almenys un empleat, d'una categoria o altra).

6.1.5. Associacions alternatives

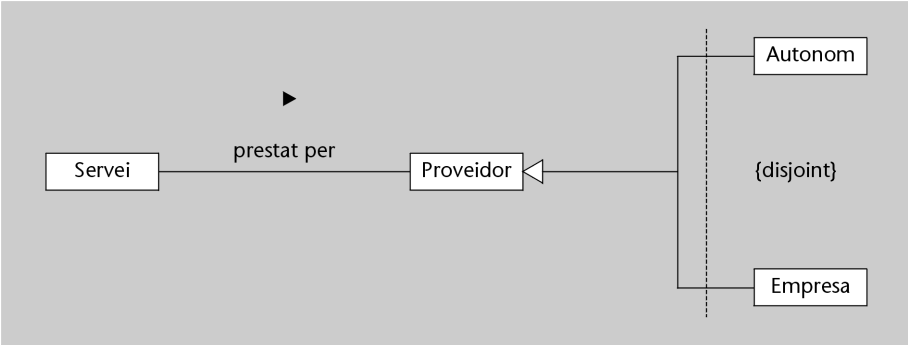
Pot passar que en una classe que participi en dues associacions, cada objecte concret participi en una o en l'altra, però no pas en totes dues. Aleshores es parla d'*associacions alternatives*.

Exemple d'associacions binàries alternatives

Una forma de representar les associacions binàries alternatives es pot veure en la figura:



Un servei determinat pot ser prestat per un proveïdor autònom o bé per una empresa, però mai per un i l'altra alhora. Això també es podria descriure així:

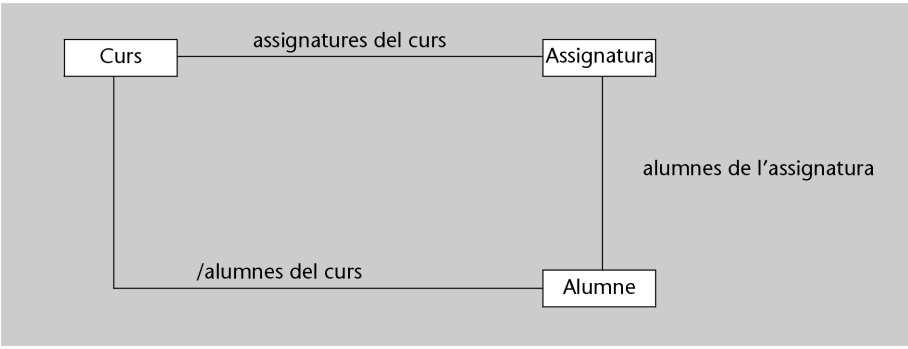


6.1.6. Associacions derivades

Una associació derivada és una associació redundant que es pot obtenir com a combinació d'altres relacions del model.

Exemple d'associació derivada

En la figura representem l'associació derivada següent:



on *alumnes del curs* és una associació derivada, perquè els alumnes d'un curs es poden determinar recorrent les assignatures del curs i trobant els alumnes de cadascuna. Però podria ser, també, que aquesta relació es definís d'una manera no redundant, per exemple que comprengués els alumnes que estiguessin matriculats només d'assignatures del curs en qüestió; llavors no es posaria "/", que és l'indicador d'element derivat, com hem vist en parlar dels atributs derivats en el subapartat 4.1.2.

6.2. Agregacions i composicions

En aquest subapartat considerem les agregacions i les composicions.

6.2.1. Agregacions

Una agregació és un cas particular d'associació binària en la qual un dels papers té el significat de 'part' i l'altre el de 'tot', en algun sentit. Anomenarem *components*, la classe corresponent al primer paper i als seus objectes, i *classe* i *objectes agregats*, els del segon paper.

Es diu que una agregació és una relació *is-part-of*.

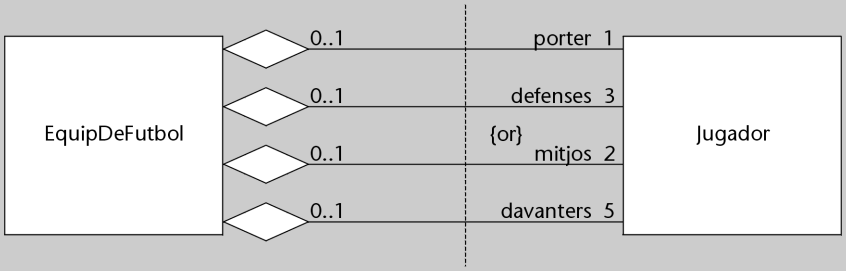
Les **agregacions** poden tenir diversos significats:

- **Acoblament-peces**: una màquina i les seves peces, un circuit elèctric i els seus components, un sistema de programari i els seus programes; cada part té un paper concret i no es pot canviar per qualsevol altra.
- **Continent-contingut**: un avió i els passatgers que transporta, que no constitueixen l'avió, ja que un avió sense passatgers és tanmateix un avió sencer.
- **Col·lectiu-membres**: una colla excursionista i els seus excursionistes, o bé una promoció d'alumnes i aquests alumnes; se suposa que els membres no tenen papers diferenciats i, per tant, són intercanviables.

Un objecte compost pot tenir objectes components que pertanyen a diverses classes; o dit d'una altra manera, una classe pot tenir relacions d'agregació amb altres de diferents. Una classe pot fer el paper de classe composta en una agregació i de classe component en una altra, i també pot fer els dos papers a la mateixa agregació (si bé un objecte no pot ser component de si mateix). Entre dues classes hi pot haver més d'una agregació.

Exemple d'agregacions

A continuació es representa la composició clàssica d'un equip de futbol:



Cada jugador o bé juga en un equip (el del club al qual pertany) o bé en cap, i un jugador només pot estar en una de les quatre posicions. Se suposa que els defenses, per exemple, són intercanviables entre ells, altrament hi hauria onze agregacions diferents. El rombe buit a la banda de la classe composta o objecte compost denota que és una agregació.

6.2.2. Composicions i objectes compostos

La composició (o agregació de composició) és un cas particular de l'agregació.

En una composició els objectes components no tenen vida pròpia, sinó que, quan es destrueix l'objecte compost del qual formen part, també es destrueixen; a més, un objecte component només pot formar part d'un objecte compost i no pot passar d'un objecte compost a un altre. Aquestes restriccions no existeixen en el cas d'agregacions en general. En una composició anomenarem la classe agregada, **classe composta**, i a l'objecte agregat, **objecte compost**.

En l'exemple d'agregacions...

.. que hem vist al subapartat anterior, els jugadors poden passar d'un equip a un altre i existeixen per si mateixos fins i tot quan no estan en cap equip; per tant, les agregacions en qüestió no són composicions.

Exemples de composició i d'objecte compost

1) Exemple de composició

Considerem la composició següent:

Un centre universitari (facultat, etc.) pertany a una sola universitat (de fet, per la banda de la classe composta la cardinalitat en una composició és sempre 1). Una universitat ha de tenir almenys un centre universitari. Suposem que si desapareix una universitat desapareixen també els seus centres universitaris i que aquests no passen d'una universitat a una altra.

2) Exemple d'objecte compost

Com que en una composició cada objecte component només pot pertànyer a un sol objecte compost, podem representar els objectes components dins l'objecte compost, en un compartiment especial.

6.3. Relacions de dependència

Una relació de dependència expressa que un element del model –anomenat **client**– depèn per a la seva implementació o funcionament d'un altre element –anomenat **subministrador***.

La relació de dependència es diu que és una relació *depends_on*.

*** En anglès, *supplier*.**

El símbol d'una relació de dependència és una fletxa de línia discontínua i punta oberta.

Hi ha diversos estereotips estàndard, algun dels quals ja hem vist, i se'n poden definir més. Els estereotips estàndard són els següents:

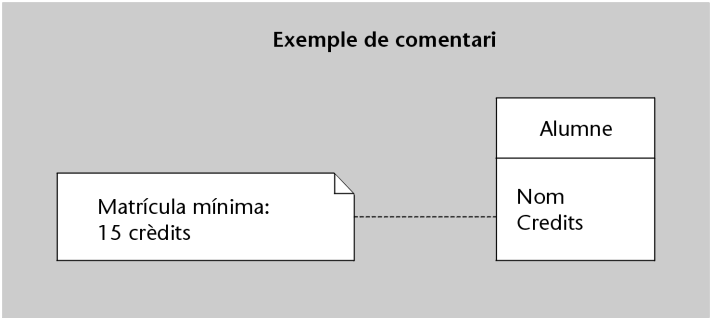
- *access*, *bind*, *import*, *instantiate*, *uses*, que ja hem vist.
- *derive* significa que un element s'obté d'un altre per mitjà d'un càlcul o algorisme.
- *friend* dóna accés al client als elements de visibilitat *private* continguts en el subministrador.
- *refine* vol dir que el client procedeix històricament del subministrador, del qual és una versió nova o enriquida (per exemple, una classe descrita a l'anàlisi a la qual es fan canvis en el disseny);
- *trace* relaciona elements que corresponen des d'un punt de vista semàntic al mateix concepte, com per exemple, un element i la seva implementació;
- *call*, *create* i *send*.
- *extend* i *include*, que existeixen només entre casos d'ús.

Vegeu les relacions entre casos d'ús al subapartat 2.3 del mòdul "UML (II): el model dinàmic i d'implementació" d'aquesta assignatura.

7. Comentaris i restriccions

7.1. Comentaris

Un comentari es posa dins un rectangle amb un vèrtex doblegat, enllaçat per una línia discontínua a l'element al qual es refereix.



7.2. Restriccions

Les restriccions* expressen condicions que ha de complir l'element del model al qual van associades. Es representen igual que els comentaris, llevat que van entre claus {}, cosa que indica que poden ser interpretades per les eines CASE.

*** En anglès, *constraints*.**

Les especificacions d'UML inclouen un llenguatge per a la descripció de les restriccions, anomenat OCL, però es pot fer servir UML sense fer servir aquest llenguatge.

OCL és la sigla d'*Object Command Language*.

7.2.1. Les restriccions de les operacions: la programació per contracte

En UML hi ha tres tipus de restriccions relatives a les operacions: precondicions, postcondicions i invariants.

- Les **precondicions** són restriccions que s'han d'acomplir abans d'executar una operació; el seu acompliment ens garanteix que l'operació s'executa partint d'un estat correcte del sistema.

- Les **postcondicions** es comproven en acabar l'execució d'una operació, i garanteixen que en acabar l'operació el sistema torni a estar en un estat correcte.

- Els **invariants** són condicions que s'han d'acomplir en tot moment; s'han de comprovar a l'inici de qualsevol operació –excepte els constructors– i en acabar.

Les restriccions poden servir per a dissenyar amb vista a fer **programació per contracte**, que es basa en unes condicions damunt d'operacions i objectes anomenades *assercions* que es poden expressar en forma de restriccions d'UML.

Resum

En aquest mòdul hem estudiat el model estàtic d'UML. S'han descrit els diferents elements del model i de cadascun se n'ha descrit la notació. Aquests conceptes han estat els següents:

- Les classes, amb les seves variants (classes abstractes, metaclasses i altres) i conceptes relacionats amb la classe (classificador, interfície, plantilla) i l'herència.

- Les relacions entre classes (associacions, agregacions, relacions de dependència).

- Els comentaris i les restriccions.

Activitats

1. Un grup de dades pot estar format per dades elementals i/o grups de dades. Representeu-ho gràficament.

2. En el departament comercial d'una empresa hi ha venedors i de cada venedor es tenen les xifres de les seves vendes en cadascun dels darrers sis mesos. Representeu-ho de dues maneres diferents.

3. Representeu una classe *Alumne* amb el nombre d'alumnes i una operació per a comptar-los.

4. En l'exemple d'agregacions, suposem que hi ha jugadors que només poden fer de porters, mentre que la resta de jugadors pot estar en qualsevol posició llevat de la de porter. Com altera això el diagrama?

Vegeu l'exemple d'agregacions al subapartat 6.2.1 d'aquest mòdul didàctic.

5. Una empresa de serveis d'informàtica pot tenir diversos projectes amb un mateix client, però cada projecte és d'un sol client; cada projecte té número i descripció, i els clients tenen NIF i denominació. Té dues menes d'empleats, tècnics i caps de projecte, tots amb NIF i nom; a cada projecte hi ha un cap de projecte i almenys un tècnic; un cap de projecte pot estar en diversos projectes, però cada tècnic pot estar com a màxim en un sol projecte. Feu-ne el diagrama estàtic.

Exercicis d'autoavaluació

1. Quan es fa una generalització, la superclasse serà sempre una classe abstracta?

2. Quina relació hi ha entre classe abstracta i classe diferida?

3. Com s'indica que una classe és abstracta?

4. Quins són els símbols i denominacions de les tres opcions estàndard de la visibilitat en UML?

5. Quins són els tres compartiments estàndard del símbol de classe?

6. Com se sap si un atribut o operació és de classe o bé d'instància?

7. Què vol dir que hi hagi "/" davant del nom d'un atribut?

8. Si entre dues interfícies hi ha una relació d'especialització, cal que estiguin implementades per dues classes, l'una subclasse de l'altra?

9. Compareu els conceptes de *superclasse* i *metaclasse*.

10. En un diagrama, com es distingeix una composició d'una agregació en general?

11. Com es representa gràficament la relació entre una metaclasse i una classe seva?

12. En una associació, quina diferència hi ha entre el sentit de l'associació i el sentit de la navegació?

13. Una relació ternària és equivalent a tres relacions binàries en triangle?

14. En l'exemple de les universitats i els centres universitaris, què passaria si els centres poguessin passar d'una universitat a una altra?

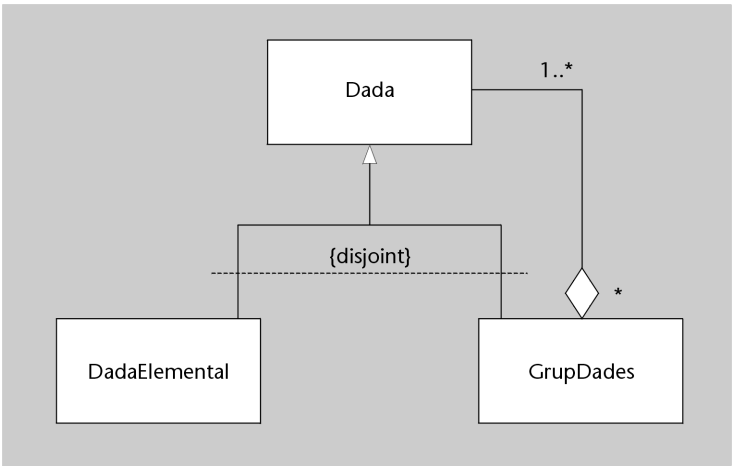
15. Com es distingeix un comentari d'una restricció?

16. Resumiu què signifiquen els diferents tipus de fletxes i línies que hi pot haver entre dues classes.

Solucionari

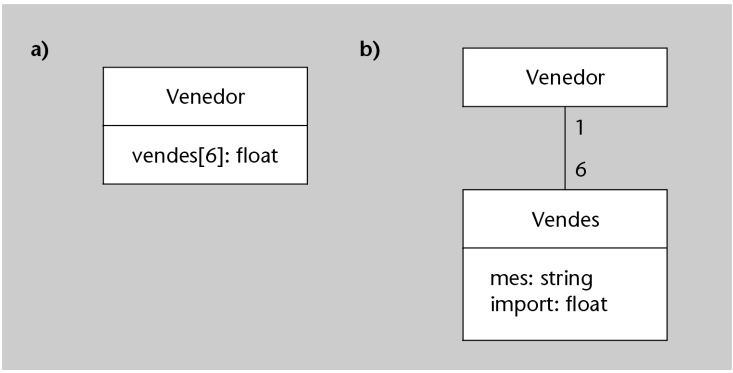
Activitats

1.

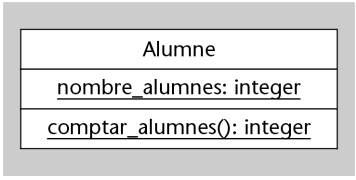


S'ha suposat que una dada pot figurar en diversos grups de dades.

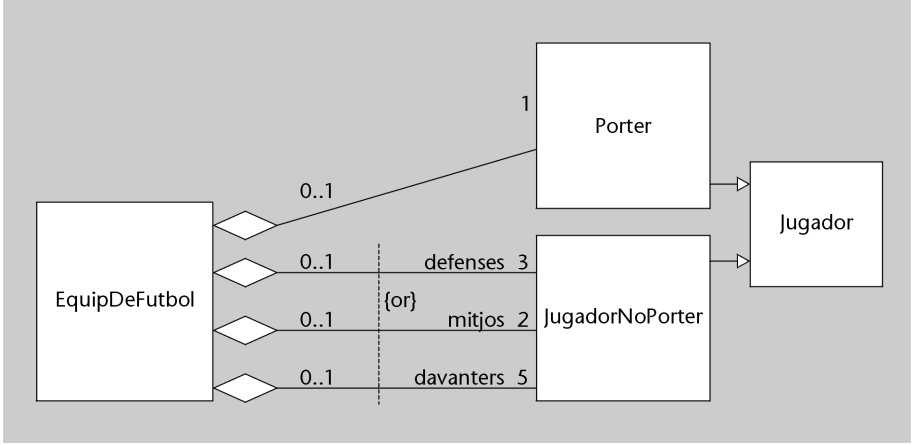
2.



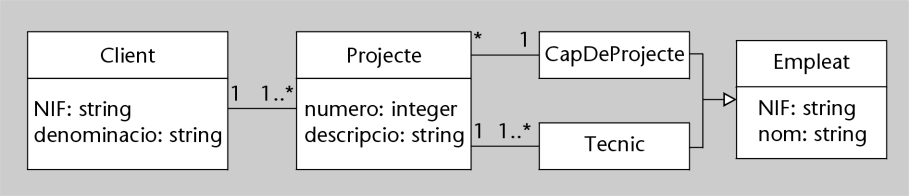
3.



4.



5.



Exercicis d'autoavaluació

1. No necessàriament, perquè pot ser que la superclasse comuna a les dues subclasses tingui també objectes que no pertanyin a cap de les dues.

2. Tota classe diferida és abstracta perquè no és instanciable; però hi pot haver classes abstractes que no siguin diferides, sinó que s'hagin obtingut per generalització a partir de les seves subclasses.

Vegeu l'herència per generalització, dins el subapartat 7.1 d'aquest mòdul didàctic.

3. Hi ha dues maneres alternatives de fer-ho: amb la *property string* {*abstract*} i posar en cursiva el nom de la classe.

4. privat: "-", protegit: "#" i públic: "+" .

5. El del nom, el dels atributs i el de les operacions.

6. Si són de classe, la seva especificació està subratllada.

7. Que és un atribut derivat.

8. No; no cal ni tan sols que estiguin implementades per dues classes diferents, n'hi hauria prou amb una classe que implementés totes les operacions de la interfície subclasse, ja que les operacions de l'altra interfície han d'estar entre aquestes.

9. Els objectes d'una metaclasse són classes, les quals tenen objectes; per tant, en una metaclasse hi ha objectes de dos nivells d'objectes; en canvi, les subclasses no són objectes de la superclasse, sinó que els objectes de les subclasses pertanyen també a la superclasse. Les classes d'una metaclasse, com que són objectes, es poden crear i destruir, mentre que les subclasses d'una superclasse són fixes.

10. En una agregació el rombe és buit, mentre que en una composició és ple.

11. Per mitjà d'una relació *instantiate*, ja que la classe és un objecte de la metaclasse.

12. El sentit de l'associació té un valor semàntic; per exemple si tenim dues classes, *A* i *B*, relacionades per l'associació *pertany a*, el sentit de l'associació ens diu si és *A* que pertany a *B*, o a l'inrevés. La punta de fletxa que indica navegació en un sentit vol dir que per mitjà de la relació es pot accedir als objectes de la classe de la punta de la fletxa des dels de la cua; hi pot haver navegació en els dos sentits, mentre que el sentit de l'associació sempre és únic.

13. No, perquè una associació ternària entre les classes *A*, *B* i *C* a un parell d'objectes, un d'*A* i l'altre de *B*, li fa correspondre un o més objectes de *C*, i això no és cap associació binària. El mateix passa si permutem cíclicament els noms de les classes.

14. Que ja no es tractaria d'una composició, sinó d'una agregació en general.

15. Que les restriccions van entre claus, cosa que es pretén que eventualment les pugui fer interpretables a l'eina que fa el diagrama.

16. Les classificarem primer per tipus de línia i després per tipus de punta de fletxa. El significat indicat correspon al sentit de la fletxa.

- **Línia contínua:**

– Punta triangular buida: subclasse de classe.

– Punta oberta: navegació per mitjà d'una associació o agregació.

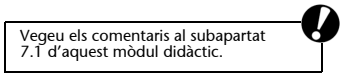
– Sense punta: associació.

- **Línia discontínua:**

– Punta triangular buida: la classe implementa la interfície.

– Punta oberta: una classe té relació de dependència respecte a l'altra.

– Sense punta: connecta una classe associativa amb l'associació corresponent (i també enllaça un comentari a l'element de diagrama al qual s'aplica).



- **Sense línia:**

– Punta triangular plena: indica el sentit d'una associació, és a dir, el paper d'una classe respecte a l'altra.

Glossari

associació binària
Aquella en què participen només dues classes.

associació derivada
Associació redundant que resulta de la combinació d'altres relacions.

associació reflexiva
Associació binària d'una classe amb si mateixa.

atribut de classe
Atribut que té un únic valor per a tota la classe.

atribut derivat
Atribut el valor del qual es pot obtenir a partir del d'altres atributs.

cardinalitat
En una associació binària, la cardinalitat d'un paper *A* és el nombre d'objectes de l'altre paper *B* a què pot estar enllaçat cada objecte d'*A*; s'indica el valor màxim i mínim d'aquest nombre. En una associació ternària és el següent: la cardinalitat del paper *A* expressa els límits al nombre d'objectes d'*A* que poden estar enllaçats a cada combinació concreta d'un objecte del paper *B* i un del paper *C*.

classe
Conjunt d'objectes que tenen els mateixos atributs i operacions.

classe abstracta
Superclasse de la qual no es poden instanciar objectes perquè tot objecte seu ha de pertànyer a alguna de les seves subclasses.

classe associativa
Associació que, pel fet de tenir atributs i/o operacions propis, esdevé una classe.

classe diferida
Classe abstracta que té alguna operació abstracta.

classe d'utilitat
Recurs per a agrupar processos i/o dades en forma d'una classe que no pot tenir objectes.

classe parametritzada
Especificació d'una classe a la qual romanen sense concretar alguns aspectes de les seves operacions o atributs en forma de paràmetres als quals s'ha de donar un valor.

classe terminal
Classe de la qual no es permet que es defineixin subclasses.

classe virtual
Vegeu *classe diferida*.

classificador
Concepte que comprèn les classes, les interfícies i els tipus de dades.

domini
Porció del món real considerada per una aplicació.

especialització
Identificació d'una o més subclasses dins una classe.

estereotip
Variant d'un element d'UML. Hi ha estereotips estàndard, definits dins UML, i se'n poden definir d'específics per a un projecte. Els estereotips s'identifiquen per una paraula clau entre « ».

generalització
Definició d'una superclasse per abstracció dels atributs i operacions comuns a diverses classes, que passen a ser les seves subclasses.

herència
Característica de les subclasses de tenir almenys tots els atributs i operacions de cadascuna de les seves superclasses.

instància
Un objecte és una instància de la seva classe. Aquest concepte s'estén als classificadors.

interfície
Conjunt d'operacions d'una classe visibles des d'altres classes.

model estàtic
Model que descriu les propietats permanents del programari en termes de classes i objectes.

operació abstracta
Operació d'una superclasse que no està implementada en aquesta sinó sols a les seves subclasses.

paquet
Element del model que pot contenir elements de qualsevol tipus, inclosos paquets. Serveix per a fraccionar el model segons algun criteri.

plantilla
Vegeu *classe parametritzada*.

visibilitat
Propietat de diversos elements del model que representa que són reconeguts en un àmbit més o menys gran extern a aquell en el qual s'han definit.

Bibliografia

Booch, G.; Rumbaugh, J.; Jacobson, I. (1999). *El lenguaje unificado de modelado*. Madrid: Addison-Wesley.

Eriksson, H.E.; Penker, M. (1997). *UML Toolkit*. John Wiley & sons.

Fowler, M.; Scott, K. (1997). *UML Distilled*. Reading: Addison-Wesley Longman.