Orientació a objectes

Jordi Pradel Miquel
Jose Raya Martos

PID_00171145

Universitat Oberta de Catalunya
www.uoc.edu

Índex

Introducció

Tal com hem vist al mòdul "Introducció a l'enginyeria del programari", els programadors necessiten un llenguatge de programació que els permeti escriure programes en termes de més alt nivell d'abstracció del que la màquina pot executar per si mateixa. Un dels tipus de llenguatges de programació més importants són els orientats a objectes.

L'orientació a objectes proposa fer una abstracció del problema que s'està intentant resoldre en lloc de fer-la de la màquina que finalment executarà el programa. Gràcies a aquest enfocament, resulta un paradigma molt útil no solament per a la programació sinó també per a l'anàlisi.

Al llarg d'aquest mòdul estudiarem com podem crear una descripció del món real seguint el paradigma de l'orientació a objectes i com aquesta descripció ens pot ajudar a fer anàlisi de requisits però també en altres activitats del programari.

Començarem introduint l'orientació a objectes i el context històric en el qual va néixer.

A continuació anirem veient, de manera progressiva, com podem fer servir el paradigma de l'orientació a objectes per a descriure el món real. Començarem parlant de la classificació i l'abstracció, mitjançant les quals estructurem la informació en el que anomenarem *classes*, *associacions*, *atributs* i *operacions* i continuarem estudiant l'encapsulament, que ens permet ocultar informació dins els objectes, i l'herència i el polimorfisme, que ens permeten fer abstraccions més potents.

Finalment veurem com podem aplicar l'orientació a objectes per a fer una descripció del món real per mitjà d'un cas pràctic.

Objectius

L'objectiu principal d'aquest mòdul és presentar el paradigma de l'orientació a objectes i com es pot aplicar a activitats de l'enginyeria del programari diferents de la programació. En concret, l'estudiant ha d'assolir els objectius següents:

1. Entendre els conceptes fonamentals de l'orientació a objectes: classificació, abstracció, ocultació d'informació, encapsulament, herència i polimorfisme.

2. Saber com es poden descriure els objectes d'un sistema mitjançant l'ús de classes, atributs, operacions i associacions.

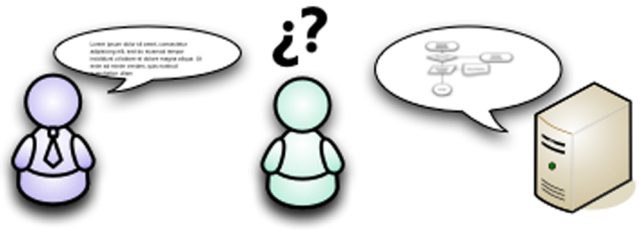
1. Què és l'orientació a objectes?

Un dels problemes comuns a tots els projectes de desenvolupament de programari és descriure la tasca que s'ha de dur a terme en un llenguatge que tant el programador com la computadora puguin entendre. Els llenguatges de programació es van crear justament per aquest motiu.

Vegeu també

Al mòdul "Introducció a l'enginyeria del programari" es tracten els diversos problemes comuns a tots els projectes de desenvolupament de programari.

El problema de la comunicació



Al món de la programació informàtica s'ha intentat solucionar el problema creant nous llenguatges de programació més propers al llenguatge natural (el que fem servir les persones per a comunicar-nos entre nosaltres) i a la manera que tenim d'entendre la realitat que ens envolta.

Exemple de llenguatge assemblador

A continuació teniu un exemple de programa en llenguatge assemblador. No us preocupeu, que no cal entendre'l per a entendre la resta del mòdul.

```
 .MODEL Small
 .STACK 100h
 .DATA
msg db 'Hello, world!$'
 .CODE
start:
mov ah, 09h
lea dx, msg
int 21h
mov ax,4C00h
int 21h
end start
```

A continuació teniu un exemple del mateix programa, aquest cop escrit en C.

```
#include<stdio.h>
int main(int arg_count,char ** arg_values) {
 printf("Hello World\n");
 return 0;
}
```

I a continuació en BASIC.

```
10 PRINT "Hello World"
```

1.1. Evolució històrica dels llenguatges de programació

Ja sabem que els processadors poden entendre un conjunt limitat d'instruccions (l'anomenat *codi màquina*). Els primers llenguatges es limitaven a instruccions molts senzilles que es corresponien pràcticament una a una amb les operacions que podia entendre el processador. Els llenguatges que implementen aquest paradigma es coneixen com a *llenguatge assemblador*.

Els llenguatges fets en assemblador són molt fàcils d'interpretar per a la màquina, però són molt complicats d'escriure i d'entendre per a les persones. Per aquest motiu, programar fent servir aquests llenguatges es fa molt feixuc i aquest enfocament només és viable per a tasques molt concretes.

De seguida es van anar elaborant llenguatges més rics que permetien als programadors fer servir un llenguatge més proper al llenguatge natural a l'hora de descriure les instruccions que ha de seguir el computador. És el cas de llenguatges com el BASIC, el COBOL o el C.

Aquests llenguatges es coneixen com a *llenguatges d'alt nivell* perquè permeten expressar-se a un nivell d'abstracció més alt que els llenguatges assembladors. També es coneixen com a *llenguatges procedimentals* perquè es basen en la definició de procediments per a descompondre el problema que es vol tractar en unitats més petites.

Els llenguatges procedimentals eleven el nivell d'abstracció respecte als llenguatges assembladors creant una abstracció de la màquina sobre la qual s'executen, que entén instruccions més abstractes que les que realment pot executar.

1.2. Els llenguatges de programació orientats a objectes

Al final dels anys seixanta els investigadors noruecs Ole-Johan Dahl i Kristen Nygaard, mentre treballaven en un programari de simulació de vaixells, van decidir provar un enfocament diferent. En comptes de fer una abstracció de la màquina sobre la qual s'executaria el programa farien una abstracció del problema sobre el qual estaven treballant. Els seus programes, en comptes de tenir procediments i estructures de dades tindrien objectes, cadascun dels quals representaria un objecte del món real; així naixia el llenguatge Simula 67 i la programació orientada a objectes.

No va ser, però, fins al final dels setanta i començaments dels vuitanta que el paradigma de l'orientació a objectes es va popularitzar massivament, primer amb el llenguatge Smalltalk (creat per un grup d'investigadors dirigit per Alan Kay al mític laboratori de Xerox a Palo Alto), després amb el C++ (creat per

Bjarne Stroustrup als no menys mítics laboratoris Bell d'AT&T el 1983) i finalment amb el Java, com un dels llenguatges més populars actualment (creat per James Gosling a Sun Microsystems el 1995).

La diferència fonamental dels **llenguatges orientats a objectes** respecte als llenguatges procedimentals està en el fet que, en comptes de basar-se en allò que la màquina pot entendre, es basen en allò que les persones volem comunicar.

Aviat es va veure, però, que el fet que aquest paradigma se centrés en l'àmbit del problema per resoldre i no pas en el de la solució, el convertia en una bona eina no solament per a les tasques d'implementació, sinó també durant l'anàlisi, ja que l'orientació a objectes ens ofereix una manera de fer la descripció de qualsevol sistema fàcilment traslladable a programari mitjançant els llenguatges de programació orientats a objectes.

1.3. Anàlisi orientada a objectes

L'anàlisi orientada a objectes consisteix a aplicar els conceptes i idees dels llenguatges orientats a objectes a l'anàlisi de sistemes. Això ens permetrà fer servir un llenguatge similar durant les diferents etapes del desenvolupament.

Aquest ús de l'orientació a objectes durant les tasques d'anàlisi es va popularitzar, especialment, a partir de l'aparició el 1997 del llenguatge unificat de modelització (UML) creat per James Rumbaugh, Grady Booch i Ivar Jacobson.

Comunicació amb UML

2. Classificació i abstracció

Un **sistema orientat a objectes** està format per una sèrie d'elements autònoms (els **objectes**) que es comuniquen els uns amb els altres per tal de dur a terme les seves tasques.

Per tant, el primer que hem de saber en un sistema com aquest és què pot fer cadascun d'aquests objectes: quines coses sap, amb quins altres objectes es pot comunicar o quines tasques se li poden demanar.

Anomenarem **responsabilitats** de l'objecte aquestes característiques.

2.1. Classificació

Quan descrivim un determinat sistema fent servir orientació a objectes descriurem els objectes que formen aquest sistema. Si intentem descriure un per un aquests objectes, però, aviat trobem que hi ha objectes que són d'un mateix tipus i tenen una mateixa estructura i responsabilitats.

Exemple de classificació

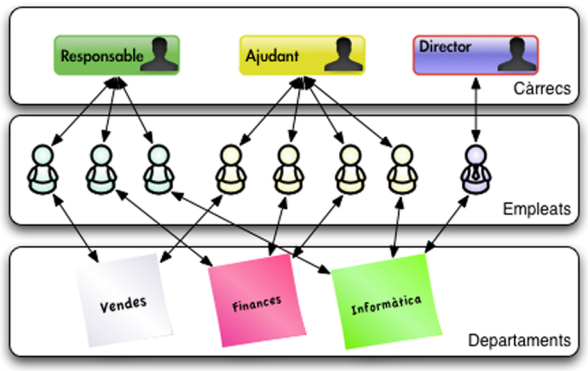
Si estem descrivint una empresa, per exemple, segurament tindrem algun objecte que representa un treballador concret. Aquest treballador pot tenir responsabilitats com "saber quin és el seu nom", "saber quin és el seu departament" o "calcular quants companys té (és a dir, quants treballadors pertanyen al mateix departament que ell)". Però de seguida veiem que hi ha molts altres objectes a la mateixa empresa que també representen treballadors i que, per tant, també tenen les responsabilitats de "saber quin és el seu nom", "saber quin és el seu departament" i "calcular quants companys té".

A l'hora de fer la descripció del nostre sistema, per tant, no ens interessarà tant l'objecte en si (en aquest cas, una persona, l'individu) com la classe d'objectes que tenen les mateixes responsabilitats (en aquest cas, els treballadors).

La **classificació** és el mecanisme mitjançant el qual simplifiquem la descripció dels objectes del sistema i identifiquem aquelles classes d'objectes tals que tots els objectes de la mateixa classe tenen les mateixes responsabilitats.

En el cas anterior, tots els treballadors saben quin és el seu nom (encara que cada treballador tingui un nom diferent, cadascú sap quin és el seu), saben quin és el seu departament i poden calcular quants companys tenen.

Classificació dels objectes



Anomenem **classe** el grup i **instància** l'objecte.

Classes d'objectes

A l'exemple anterior tenim tres classes d'objectes:

- Càrrecs,
- Empleats i
- Departaments.

De la classe *Càrrec* en tenim tres instàncies (*Director*, *Ajudant* i *Responsable*), d'*Empleat*, vuit, i de *Departament*, tres (Vendes, *Finances* i *Informàtica*).

A l'hora de definir les responsabilitats dels objectes d'una classe, les podem dividir en tres tipus:

1) La informació que coneixen (les dades)

2) Els objectes que coneixen (les associacions)

3) Les tasques que poden dur a terme (les operacions)

A l'exemple anterior, cada instància d'*Empleat* sap quina és la informació de l'empleat (el nom, el telèfon, etc.), està associada amb altres instàncies (el seu càrrec i el seu departament) i hi ha una sèrie de tasques que pot dur a terme (redactar una sol·licitud, aprovar-la, etc.).

Instàncies

2.2. Abstracció

En orientació a objectes, l'**abstracció** és el mecanisme mitjançant el qual decidim quines de les possibles responsabilitats associades a una classe formaran part de la seva definició.

A l'hora de descriure un sistema mitjançant l'orientació a objectes, haurem d'aplicar el mecanisme d'abstracció als tres tipus de responsabilitat que hem vist anteriorment per a decidir quines dades, relacions i comportament són rellevants per al nostre problema.

En el cas de l'exemple anterior, és ben probable que trets com el color dels ulls, l'alçada o qui és amic de qui entre els treballadors o el fet que a alguns treballadors els poguem demanar que facin la declaració de renda no siguin rellevants i, per tant, no es tinguin en compte.

2.3. Descripció de les classes d'objectes

Aplicant la classificació i l'abstracció arribarem a una primera descripció dels objectes que formen part del nostre sistema, en què tindrem un conjunt de classes i, per a cada classe, un conjunt de responsabilitats. Aquestes responsabilitats prendran la forma d'atributs, operacions i associacions.

2.3.1. Atributs

Quan tots els objectes d'una classe tenen un mateix element d'informació (com ara la data de contractació a l'empresa), encara que cada objecte hi tingui un valor diferent, diem que la classe té un atribut.

Els **atributs** ens diuen quina informació (quines dades) coneixem sobre els objectes d'una classe determinada.

L'atribut tindrà un **nom** que ens permet distingir-lo de la resta d'atributs (per exemple, podem anomenar *Alta* l'atribut que desa la data en què un empleat va ser donat d'alta a l'empresa) i sol tenir un **tipus** que ens ajuda a entendre millor el tipus d'informació que s'hi emmagatzema i a saber quins valors són vàlids per a l'atribut (per exemple, si diem que *Alta* és de tipus *Data*, sabem que els valors que pot prendre han de ser dates i no pas nombres enters).

String

Anomenem *String* el tipus d'atributs en que els valors són seqüències de caràcters; és a dir, textos sense cap format.

Formalment, un String és una seqüència de símbols que s'escullen d'un conjunt o d'un alfabet.

Exemple de classe amb atributs i de dues de les seves instàncies

Àmbit de l'atribut

Els atributs poden emmagatzemar, o bé un valor diferent per a cada instància, o bé un mateix valor per a totes les instàncies de la classe. Per exemple, l'atribut *Alta* que emmagatzema la data en què es va incorporar el treballador a l'empresa ha de tenir un valor diferent per a cada treballador. En aquest cas, diem que l'atribut té **àmbit d'instància**.

En canvi, un atribut anomenat *mitjanaEdats* que emmagatzemés la mitjana de les edats de tots els treballadors no tindria sentit que donés valors diferents per a diferents instàncies ja que, sigui quin sigui el treballador a qui preguntem, la mitjana de les edats ha de ser la mateixa; en aquest cas, diem que l'atribut té **àmbit de classe**. Aquests atributs (amb àmbit de classe), però, són poc habituals.

Nombre de valors d'un atribut

Fins ara hem vist atributs tals que cada instància només pot tenir un valor, i l'ha de tenir per força. Diem que són atributs univaluats i obligatoris. Però alguns atributs poden no tenir valor en algunes instàncies (atributs **opcionals**) o en poden tenir més d'un (atributs **multivaluats**).

Exemples d'atributs opcionals i multivaluats

El número de fax d'un treballador podria ser un atribut opcional (ja que un treballador podria no tenir número de fax); d'altra banda, el número de telèfon podria ser multivaluat, ja que podria ser que un treballador tingués més d'un número de telèfon en el qual es pogués localitzar.

2.3.2. Associacions

Les associacions ens diuen quins altres objectes coneixen els objectes d'una classe (quines són les seves relacions amb altres objectes). Per tant, formen part del conjunt d'informació (les dades) que gestiona un objecte i, com a tals, les podríem considerar atributs. A diferència dels atributs que hem considerat fins ara, però, una associació no fa referència a un únic objecte sinó que sempre farà referència, com a mínim, a dos objectes. Per això distingim entre atributs i associacions.

Per exemple, hem dit que ens interessa saber, per a una instància concreta de departament, quins empleats hi pertanyen. En aquest cas diríem que hi ha una associació entre les classes *Empleat* i *Departament*. L'associació pot tenir un nom (per exemple, podem anomenar *treballaA* la relació entre un empleat i el seu departament). A diferència d'un atribut, l'associació *treballaA* sempre fa referència a dos objectes: un empleat i un departament.

Una **associació** entre dues classes indica que les instàncies de cada classe saben, com a part de les seves responsabilitats, amb quina o quines instàncies de l'altra classe estan relacionades.

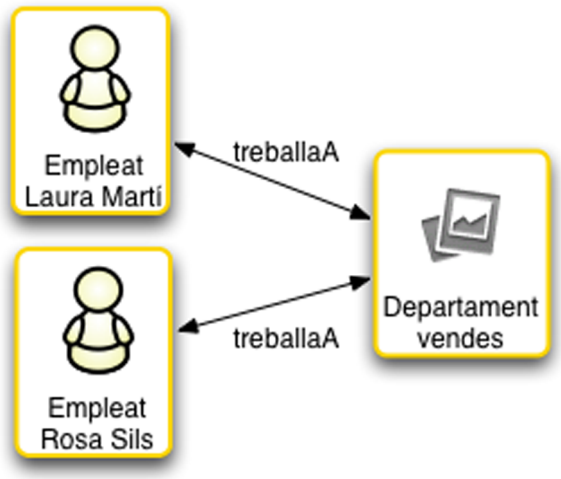
La relació d'associació entre dues instàncies és recíproca, de tal manera que si una instància *a* d'una classe *A* està associada amb una instància *b* de la classe *B*, la instància *b* estarà també associada a la instància *a*.

Quan una classe està associada amb una altra, podem consultar el vincle entre les instàncies relacionades. Així, si diem que la classe *Empleat* està associada a la classe *Departament*, estem dient que, donada una instància concreta de la classe *Empleat*, li podrem demanar amb quines instàncies de la classe *Departament* està associada o, si ho volem fer a la inversa, donada una instància concreta de *Departament*, li podrem demanar amb quines instàncies d'*Empleat* està associada.

Exemple d'associacions

A l'exemple de la figura següent, podem saber que l'empleada Laura Martí té associat el Departament de Vendes, l'empleada Rosa Sils, també, i que el Departament de Vendes té associats els empleats Laura Martí i Rosa Sils.

Associació Empleat-Departament(treballaA)



Tot i que l'associació és una característica de la classe, els valors que pren seran específics de les instàncies (igual que passa amb els atributs). Però, a diferència dels atributs, les associacions mai no tindran àmbit de classe.

Rols de les instàncies

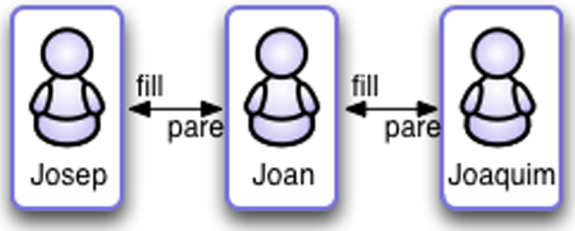
Quan una instància participa en una associació amb altres instàncies, sempre ho ha de fer tenint un rol concret. De vegades, aquest rol tindrà un nom explícit i, fins i tot, podrà tenir una visibilitat (públic, privat, etc.).

El rol és especialment important en el cas de les **associacions recursives**, que són aquelles associacions que connecten una classe amb ella mateixa.

Exemple d'associació recursiva

Si tenim la classe *Persona* i volem definir la relació entre pares i fills, tindríem una associació recursiva a la classe *Persona*, ja que estem connectant la classe *Persona* amb ella mateixa. En aquest cas, una instància de persona pot tenir el rol de pare o de fill en relació a una altra instància. És important veure, però, que una mateixa instància de *Persona* pot tenir el rol de pare en relació a una instància i, al mateix temps, tenir el rol de fill en relació a una altra.

Associació recursiva



El rol també és important quan tenim més d'una associació entre dues classes.

Per exemple, la classe *Persona* i la classe *Ciutat* podrien tenir dues associacions: la que ens diu on va néixer una persona (ciutat natal) i la que ens diu on viu la persona (ciutat de residència). En aquest cas no n'hi ha prou de dir que una ciutat està associada a una persona sinó que cal distingir quin dels dos rols té la ciutat en la seva associació amb la persona (és a dir, si és la seva ciutat natal o si és la seva ciutat de residència). *A priori*, res no impedeix que una mateixa ciutat tingui els dos rols però, tot i així, consideraríem que la ciutat està associada dues vegades a la persona, una com a ciutat natal i una altra com a ciutat de residència.

Classes associatives

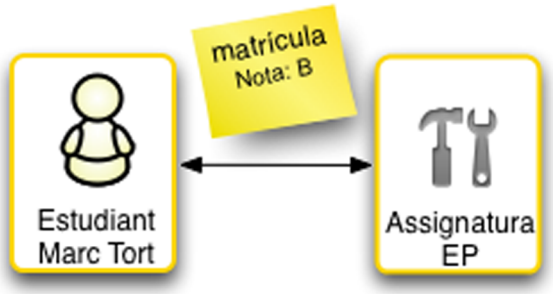
Nota d'un estudiant

Suposem el cas següent. Un estudiant està matriculat d'una assignatura i, en finalitzar el curs, li queda una nota de B. De quina classe és atribut la nota que l'estudiant treu d'una assignatura? Si fem que la nota sigui un atribut d'*estudiant* llavors no dependria de l'assignatura; diríem, per exemple, que en Joan (un estudiant) té una nota de B... però, de quina assignatura? Si, en canvi, definim la nota com un atribut d'*assignatura*, llavors no dependria de l'estudiant. En realitat, la nota pertany a l'associació entre l'estudiant i l'assignatura.

Una **classe associativa** és aquella classe les instàncies de la qual són instàncies d'una associació. A l'exemple anterior, tindríem una classe associativa anomenada *Matrícula* les instàncies de la qual serien les parelles (*Estudiant*, *Assignatura*) que estan relacionades.

Una **instància d'una associació** és una parella d'instàncies de les classes que l'associació relaciona. Podem parlar d'instàncies d'associació tant si l'associació és una classe associativa com si no ho és. En el primer cas, però, aquestes instàncies són, alhora, objectes d'una classe i, per tant, poden tenir, al seu torn, atributs, operacions, mètodes i associacions.

Classe associativa



Associacions *n*-àries

Fins ara, totes les associacions que hem vist eren entre dues classes però, de vegades, es pot donar el cas de tenir associacions entre tres o més classes.

Exemple d'associació *n*-ària

La matrícula d'un estudiant la podem veure com una associació binaria (*Estudiant*, *Assignatura*) o bé com una associació ternària (*Estudiant*, *Semestre*, *Assignatura*).

En el segon cas, no hi ha una única associació entre l'estudiant i l'assignatura sinó diverses, cadascuna corresponent a un semestre diferent.

En el cas de tenir una classe associativa, les instàncies de la classe sempre estaran relacionades amb una instància de cadascuna de les classes de l'associació. Així doncs, no té sentit parlar de la nota de l'estudiant en aquella assignatura sinó de la nota de l'estudiant en aquella assignatura i en aquell semestre.

2.3.3. Operacions

Les **operacions** ens diuen quines tasques poden dur a terme les instàncies d'una classe (quin és el seu comportament observable).

A l'hora de definir el comportament d'un objecte distingim entre l'operació (la definició de què pot fer) i el mètode (el conjunt d'instruccions que indica com ho ha de fer). Les operacions formen part de la **interfície**, mentre que els mètodes formen part de la **implementació**.

Vegeu també

Podeu trobar més informació sobre la diferència entre interfície i implementació al mòdul "Introducció a l'enginyeria del programari".

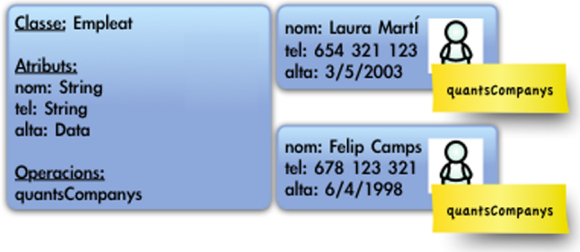
Quan un objecte vol que un altre executi el comportament definit en una de les seves operacions, diem que **invoca** (o **crida**) aquella operació o bé que li **envia** aquell **missatge**.

Per exemple, si la classe *Treballador* té una operació anomenada *quantsCompanys*, diem que s'invoca o es crida l'operació *quantsCompanys* o bé que se li envia el missatge *quantsCompanys*.

Operacions, àmbit de classe o d'instància

Les operacions també poden tenir àmbit de classe (el missatge el rep la classe) o d'instància (el missatge el rep una instància en concret) tot i que, de moment, no cal que ho tinguem en compte.

Exemple de classe amb atributs i operacions i de dues de les seves instàncies



3. Ocultació d'informació i encapsulament

Hem vist que, gràcies a la classificació i l'abstracció, podem simplificar la descripció dels objectes agrupant-los en classes i centrant-nos només en aquelles responsabilitats que siguin rellevants per al nostre cas.

També hem vist que ens interessa "ocultar informació" per tal de separar la interfície d'un objecte de la seva implementació i poder oferir a la resta d'objectes del sistema una versió encara més simplificada d'aquest.

Vegeu també

Hem vist al mòdul "Introducció a l'enginyeria del programari" que ens interessa separar la interfície d'un objecte de la seva implementació.

L'**encapsulament** és el mecanisme de l'orientació a objectes que ens permet **ocultar informació**.

3.1. Ocultació d'informació

El concepte d'ocultació d'informació és molt anterior als llenguatges orientats a objectes. De fet, va ser mencionat per primer cop per David Parnas (1972).

Ocultació d'informació segons David Parnas

Cada element (mòdul o subsistema) del sistema conté una certa informació que "oculta" a la resta de mòduls presentant-los una visió simplificada.

Una descripció en què no expliquem el funcionament intern de l'objecte descrit es coneix com a descripció de "caixa negra".

Exemple de descripció de "caixa negra"

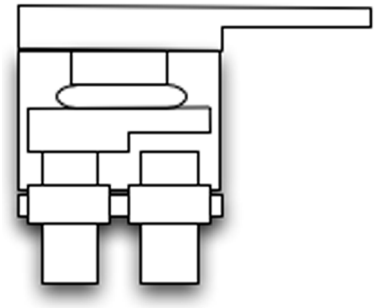
Suposem que volem descriure a una altra persona com funciona una aixeta de palanca. Segurament la nostra descripció seria de l'estil de: "L'aixeta té una palanca que, quan la puges, fa que surti aigua (com més la puges més aigua surt) i quan la baixes tanca l'aixeta. Per a regular la temperatura has de moure la palanca a l'esquerra (si vols l'aigua més calenta) o a la dreta (si la vols més freda)".

Aixeta vista com a caixa negra

Amb aquesta descripció, però, només hem descrit com es fa servir l'aixeta, no pas com funciona per dintre. Si fos un lampista qui li demana a un altre lampista com funciona una aixeta de palanca segurament ens diria alguna cosa similar al següent:

"L'aixeta té dues entrades d'aigua, una de freda i una de calenta. La palanca va connectada a una ròtula que regula el pas d'aigua de les dues entrades: a la posició central deixa passar aigua de tots dos circuits, mentre que si movem la palanca, anirem ampliant el pas d'un circuit al mateix temps que tanquem el pas de l'altre."

Aixeta vista com a caixa blanca



Les descripcions que inclouen els detalls de funcionament les coneixem com a descripcions de "caixa blanca".

Totes dues són descripcions vàlides de l'aixeta de palanca, però mentre que la primera és molt fàcil d'entendre, la segona requereix una mica més d'esforç i de coneixements sobre lampisteria. Aplicant el principi d'ocultació d'informació, la primera descripció omet el detall sobre com es regulen els dos circuits d'aigua i es limita a explicar-nos que, movent la palanca, alterem la temperatura de l'aigua.

De la mateixa manera, en descriure les responsabilitats d'una classe, ho podem fer des d'un punt de vista extern (com una "caixa negra"), indicant només com es poden fer servir els objectes d'aquella classe, o bé des d'un punt de vista intern (o de "caixa blanca"), indicant com està estructurat l'objecte i com funciona per dintre.

3.2. Encapsulament

L'**encapsulament** ens permet ocultar informació a altres objectes i definir quins atributs, operacions i associacions d'un objecte els seran visibles i quins estaran ocults.

D'aquesta manera els atributs i operacions que no hem fet visibles queden ocults a la resta d'objectes que, per tant, no necessiten (de fet no poden) saber de la seva existència. El fet que les dades i les operacions formin part de la mateixa entitat (l'objecte) facilita encara més l'ocultació d'informació.

S'anomena **visibilitat** d'un atribut, associació o operació, la propietat que defineix quins objectes el poden veure.

La visibilitat, doncs, determina quins objectes poden consultar el valor de l'atribut o de l'associació o bé invocar l'operació. En general, diem que la visibilitat determina quins objectes tenen accés a una responsabilitat determinada. Els diferents llenguatges de programació defineixen diferents tipus de visibilitat, però els casos més típics són:

Vegeu també

Veurem el concepte de subclasse en l'apartat 4 d'aquest mòdul.

- Visibilitat pública: qualsevol objecte hi té accés.
- Visibilitat privada: només els objectes de la mateixa classe hi tenen accés.
- Visibilitat protegida: només els objectes de la mateixa classe o d'alguna subclasse hi tenen accés.
- Visibilitat de paquet: només els objectes del mateix grup o "paquet" hi tenen accés.

Paquet

Un paquet és una agrupació d'elements en un sistema orientat a objectes per a facilitar-ne l'organització. Un paquet pot contenir classes i altres paquets i pot resultar útil per a definir la visibilitat de paquet dels atributs, associacions i operacions de les classes que conté.

Com a efecte col·lateral, l'ocultació d'informació té l'avantatge que algú podria crear una implementació alternativa (per exemple, fabricar una aixeta de palanca que substituís la ròtula per algun altre mecanisme) sense que això afectés els usuaris de la classe, ja que el comportament observable (la interfície) seria el mateix.

Exemple d'encapsulament

A la figura següent tenim dues possibles implementacions per a una mateixa interfície. En tots dos casos, els atributs públics són els mateixos (Nom, Telèfon, Antiguitat) però es calcularien de manera diferent a partir de conjunts d'atributs privats diferents.

Encapsulament

4. Herència i polimorfisme

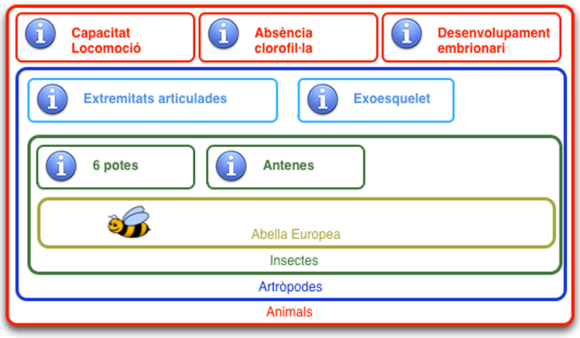
4.1. Herència

Fins ara només hem tingut en compte un nivell de classificació (un objecte pertany a una classe) però, de la mateixa manera que la classificació ens permet simplificar la descripció que fem d'un objecte agrupant els objectes en classes, de vegades podem simplificar la descripció d'una classe identificant responsabilitats comunes entre les instàncies de diferents classes.

Classificació dels éssers vius

Podem considerar el cas de la classificació dels éssers vius. Una abella qualsevol podria ser un exemplar d'abella europea (*Apis mellifera*). Però a l'hora de descriure com és l'abella, sabem que té característiques comunes amb altres éssers vius que no són abelles (els altres insectes) que, al seu temps, comparteixen característiques amb altres éssers vius que no són necessàriament insectes (com els artròpodes), que al seu temps comparteixen característiques amb la resta d'animals.

Classificació de l'abella europea



Aquest tipus de classificació jeràrquica el podem reproduir en un sistema orientat a objectes mitjançant el mecanisme d'**herència** entre classes.

Diem que una classe (**subclasse**) hereta la definició d'una altra classe (**superclasse**) quan volem reutilitzar la definició de la superclasse de manera que aquesta s'apliqui també a les instàncies de la subclasse. Aquest mecanisme ens permet, a l'hora de definir una subclasse, indicar només aquelles característiques que en són específiques, mentre que les que són comunes amb altres subclasses de la mateixa superclasse, diem que les hereta.

Tornant a l'exemple de les abelles, quan estem definint la classe *abella*, no cal que indiquem que té tres parells de potes sinó que en tenim prou de dir que és una subclasse d'*insecte* (i, per tant, la classe *abella* hereta aquesta característica). Si modifiquem la definició de la classe *insecte* i afegim que té dues antenes, automàticament totes les subclasses d'*insecte* (entre elles, *abella*) passen a heretar aquesta nova característica.

Relacions *is-a* i *is-like-a*

La relació d'herència es coneix també com a relació *is-a* (és-un), ja que les instàncies de la subclasse són també de la superclasse (una abella *is-a* un insecte). De vegades, però, ens pot interessar que la subclasse afegeixi característiques que no són aplicables a la superclasse (per exemple, una bomba de calor és com un aparell d'aire condicionat però que en comptes de només refredar també pot donar calor). En aquests casos, la relació seria *is-like-a* (és-com-un) i hi ha certa controvèrsia sobre si aquest és o no un ús correcte de l'herència.

La relació d'herència no s'ha de limitar necessàriament a un sol nivell ni a dues subclasses només, sinó que podem arribar a tenir classificacions arbitràriament complexes.

4.1.1. Generalització i especialització

Hi ha dos processos que ens permeten identificar fàcilment les relacions d'herència al nostre domini: la generalització i l'especialització. El procés de **generalització** consisteix a trobar classes més generals a partir de les que ja tenim, mentre que l'**especialització** consisteix a trobar classes més específiques a partir de les que ja tenim.

La generalització és un procés que es du a terme en dues etapes: el primer pas és identificar responsabilitats comunes a dues o més classes.

Exemple de primera etapa de la generalització

A la universitat tenim instàncies d'*Estudiant*, de les quals sabem que tenen un nom, uns cognoms, una edat i que estan matriculats d'un nombre concret de crèdits. També tenim instàncies de *Professor*, de les quals sabem que tenen un nom, uns cognoms, una edat i una antiguitat a la universitat. Podem veure, doncs, que els estudiants i els professors, si bé no són la mateixa classe (per exemple, cap professor no està matriculat de crèdits ni sabem l'antiguitat de cap estudiant), sí que comparteixen algunes característiques (nom, cognoms i edat).

El segon pas de la generalització consisteix a crear una nova classe i crear una relació d'herència entre les classes que teníem i la classe que hem afegit.

Al nostre exemple, afegiríem una classe *Persona* i faríem que, tant Estudiant com Professor, fossin subclasses de Persona.

L'**especialització** és el procés simètric a l'anterior: primer identifiquem un subconjunt de les instàncies d'una classe que comparteixen alguna característica comuna (per exemple, hi ha un subconjunt de persones que es poden matricular d'assignatures, mentre que la resta no ho poden fer) i després creem una subclasse nova que inclou aquest conjunt d'instàncies i afegeix a la seva definició la característica que els és específica.

Si estudiéssim el ADN de les abelles europees, veuríem que no són totes iguals, sinó que hi ha diversos grups, com ara el grup africà, el mediterrani o el de l'Orient Mitjà. En aquest cas, a partir de la classe *abella europea*, aniríem creant subclasses per especialització a mesura que anéssim descobrint els diferents grups.

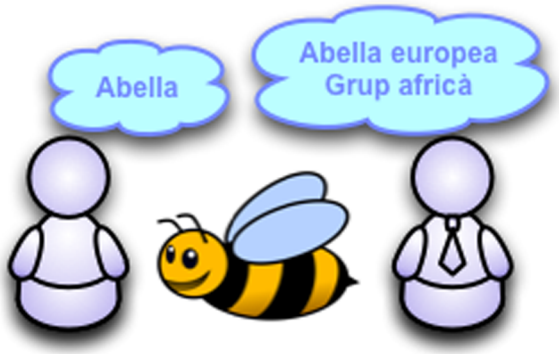
4.2. Polimorfisme

En el cas de les abelles, un mateix espècimen el podem veure com una abella (per exemple, si no hi entenem gaire) o com una abella europea del grup africà (si, per exemple, estiguéssim interessats a estudiar la distribució geogràfica dels diferents grups). Això és molt útil perquè ens permet treballar a diferents nivells d'abstracció segons quin sigui el nostre interès en els detalls.

El **polimorfisme** és la capacitat d'un objecte de presentar-se amb formes diferents (interfícies diferents) segons el context.

A l'exemple anterior, l'espècimen es presenta amb forma (o interfície) d'abella per a l'observador no expert mentre que, per a l'observador expert, es presenta en forma d'abella europea del grup africà.

Polimorfisme



És important veure, però, que l'espècimen (l'individu, la instància) és el mateix en tots dos casos i que, per tant, el polimorfisme està relacionat amb la visió que els altres objectes tenen de l'objecte més que no pas amb l'objecte en si, que sempre pertany a la classe més concreta de la jerarquia, encara que alguns observadors no ho vegin.

A l'exemple anterior, encara que l'observador no expert només vegi una abella, l'abella que està veient sempre és una abella europea del grup africà.

Exemple de polimorfisme

Si volem saber el nom d'una persona que participa al campus virtual de la UOC, ens és igual saber si és un consultor o un estudiant i, per tant, farem servir la interfície de persona independentment del tipus concret de persona que sigui. En canvi, si volem saber si la persona està matriculada d'una assignatura, hem de tenir en compte el tipus de persona que és, ja que els consultors no es matriculen i, per tant, la pregunta només té sentit per als estudiants (i hem de fer servir la interfície d'estudiant).

El polimorfisme afecta tant les dades com el comportament, i permet que les subclasses redefineixin algunes operacions per tal de comportar-se de manera diferent de la resta d'instàncies (és el que anomenem *operacions polimòrfiques*).

Una **operació polimòrfica** és una operació que té un comportament diferent en funció de la classe concreta a la qual pertanyi un objecte.

Per exemple, hi ha grups d'abelles que són més agressius que altres i que, per tant, respondran de manera diferent al mateix estímul. En aquest cas, podem dir que la classe *abella* defineix l'operació *molestar* amb el comportament *fugir* però que hi ha una subclasse que la redefineix amb el comportament *atacar*; diem que l'operació *molestar* és polimòrfica perquè si molestem una abella el seu comportament serà diferent segons el tipus concret d'abella que sigui (podria fugir si és una de les normals o atacar-nos si és una de les agressives).

4.2.1. Classes abstractes i redefinició del comportament

En una classificació jeràrquica com les que estem proposant, podria passar que una superclasse no tingués cap instància que no pertanyés a una subclasse (per exemple, la classe *abella* no té cap instància que no sigui d'alguna subclasse com ara l'abella europea).

Anomenem **classes abstractes** les classes que no es poden instanciar directament; aquestes classes s'han d'instanciar per mitjà de les seves subclasses.

A l'exemple anterior, si volem una instància d'abella, hem de decidir de quin subtipus d'abella ha de ser.

Les classes abstractes poden definir tant dades com comportament que heretaran les seves subclasses. Una característica, però, específica de les classes abstractes, és que poden definir **operacions abstractes**.

Una **operació abstracta** és una operació que no té associat un mètode (és a dir, no s'ha definit un comportament) en la classe en què es defineix.

Això obliga les subclasses a definir un mètode (comportament) per a aquella operació o a ser igualment abstractes i deixar que siguin les seves subclasses les que defineixin el comportament.

Exemple d'operació abstracta

Si tornem a l'exemple de les aixetes, podem considerar que tenim una classe abstracta *Aixeta* i dues subclasses, *Aixeta de palanca* i *Aixeta termostàtica*. En aquest cas, la classe *Aixeta* pot oferir una operació anomenada *tancar*, que tancarà l'aixeta i farà que en deixi de brollar aigua.

El comportament d'aquesta operació, però, serà diferent en funció del tipus concret d'aixeta. En aquest exemple, l'operació *tancar* de la classe *Aixeta* és abstracta perquè no defineix cap implementació definida i les subclasses *Aixeta de palanca* i *Aixeta termostàtica* la implementen de manera diferent (cadascuna definint el seu comportament propi).

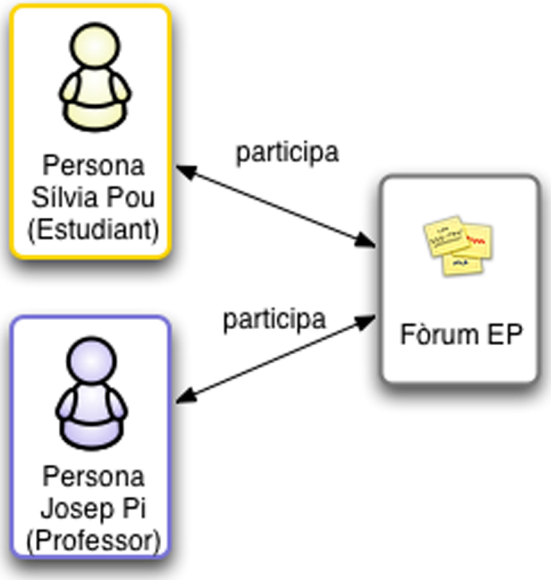
4.2.2. Associacions polimòrfiques

Les associacions ens permeten donar una nova utilitat al polimorfisme: podem tenir associacions amb una superclasse de manera que no ens hem de preocupar de la subclasse concreta dels objectes que tenim associats.

Exemple d'associació polimòrfica

Un aula del campus virtual pot tenir una llista de persones participants, per a les quals no necessita saber si són estudiants o professors, sinó només saber si són persones, si participen a l'aula i quin és el seu nom.

Associació polimòrfica



Aquest ús del polimorfisme, a més, ens ajudarà a evolucionar el sistema si, en un futur, descobrim altres tipus de persones (sempre que aquestes també tinguin nom i puguin rebre missatges).

5. Cas pràctic: un fòrum virtual

A continuació aplicarem els principis i les tècniques vistos anteriorment a un domini molt petit però prou representatiu: un sistema de fòrums virtuals en què els usuaris envien missatges a un fòrum i poden llegir els missatges que han enviat els altres usuaris.

És molt important, però, abans d'aplicar el què hem vist, que considerem quina és la finalitat de la tasca que estem duent a terme. Com hem dit amb anterioritat, el paradigma de l'orientació a objectes es pot aplicar en àmbits diferents, com ara la descripció del domini durant l'anàlisi o bé la implementació del programari del sistema. En aquest exemple ens centrarem en l'ús de l'orientació a objectes com a tècnica per a la descripció del domini del problema i, per tant, el nostre objectiu és identificar les classes del domini i no pas les classes de programari que formaran part de la solució. Per tant, veurem classes com *Missatge* o *Fòrum* però no pas *BaseDeDadesDeMissatges*.

Observació

El model del domini que fem durant l'anàlisi fa servir els objectes per a representar conceptes del món real i, per tant, no ens interessa identificar-hi operacions ni assignar-los mètodes, ja que aquestes tasques es faran durant el disseny i implementació del sistema.

5.1. Classificació dels objectes

Començarem identificant les classes i els atributs dels objectes. En aquest primer pas crearem les abstraccions bàsiques del nostre sistema i descartarem tots els detalls que no considerem rellevants.

5.1.1. Classes

En aquest cas, fent un exercici d'abstracció, podem identificar fàcilment tres classes ben diferenciades: *Usuari*, *Missatge* i *Fòrum*. Usuaris són aquelles persones que fan servir el sistema, missatges són el que s'envien els usuaris, i els fòrums serveixen per a agrupar els missatges.

Tots els objectes que tinguem al domini pertanyeran a alguna d'aquestes tres classes i, per tant, no considerarem cap classe més si no és que incorporem més informació que la que tenim actualment.

Classes

5.1.2. Atributs

Per a determinar els atributs de les diferents classes, hem de determinar quins detalls són rellevants per al nostre sistema de les instàncies de cada classe.

Dels usuaris podríem conèixer molta informació (el nom, l'edat, les seves aficions, el color dels ulls, on treballen, si els agrada l'òpera, etc.), però molta d'aquesta informació no és rellevant per al sistema que estem desenvolupant i, per tant, no l'hem de considerar com a atributs de la classe.

En canvi, segur que necessitarem una manera d'identificar els usuaris (per exemple, fent servir un nom que podria ser el seu nom real o un àlies) per tal de poder saber qui ha creat un missatge i, de la mateixa manera, necessitarem poder determinar que un usuari és qui diu ser (per exemple, mitjançant un mot de pas); en aquest cas, doncs, diríem que la classe *Usuari* té dos atributs: *nomUsuari* (el que fa servir per a autenticar-se al sistema) i *motDePas* (la contrasenya).

Altres detalls de l'usuari, com ara el nom, els cognoms i l'edat, podrien ser atributs o no segons els considerem rellevants per al sistema que estem descrivint. Per simplificar l'exemple, els considerarem no rellevants i, per tant, no els convertirem en atributs.

Dels missatges també en podríem saber tot de coses com ara la longitud del contingut, si és interessant o és avorrit, etc., però ens quedarem amb dos atributs que considerem rellevants: títol i contingut.

Finalment, dels fòrums ens interessarà saber-ne el nom (per poder-los distingir dels altres).

Per tant, tenim les classes i atributs següents:

Classes i atributs

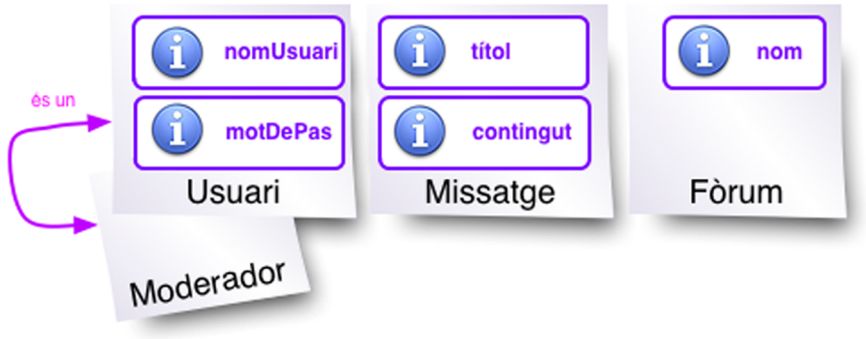
5.2. Herència

En aquest cas no és possible generalitzar cap de les classes identificades, ja que representen conceptes molt diferents entre si. En canvi, podria passar que una anàlisi més a fons de la funcionalitat ens portés a pensar que alguns dels usuaris tenen permisos especials i poden esborrar els missatges d'altres usuaris.

Com que no tots els usuaris tenen aquest permís especial, podríem identificar un subconjunt d'Usuaris que anomenaríem *Moderadors*, que són aquells que poden esborrar missatges d'altres usuaris.

En aquest cas, doncs, hauríem afegit una classe nova per especialització:

Especialització de Moderador



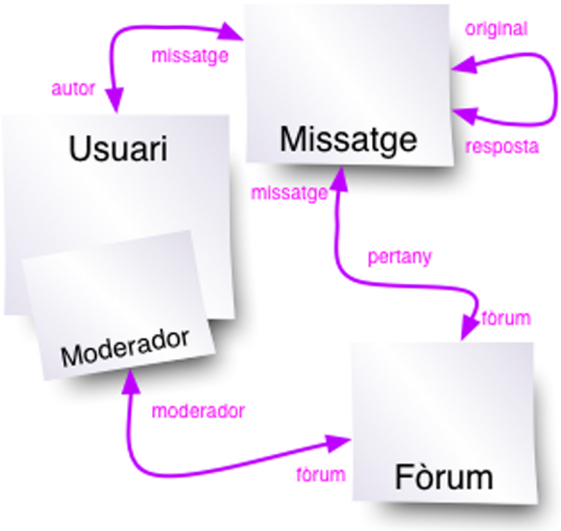
5.3. Associacions

Pel què fa a les associacions, podríem identificar una associació entre Usuari i Missatge, en què Usuari té el rol d'autor i Missatge el de missatge, i on desarem quin usuari és l'autor d'un missatge. També necessitarem saber a quin fòrum pertany un missatge (anomenarem *pertany* aquesta associació i *missatge* i *fòrum*, els rols del missatge i del fòrum, respectivament).

Alguns missatges poden ser respostes a missatges anteriors. Aquesta associació és una associació recursiva, ja que associa dues instàncies de la mateixa classe (Missatge): una té el rol de missatge original i l'altra té el rol de resposta.

Finalment, també ens interessarà saber qui és el moderador d'un fòrum en concret. En aquest cas, en comptes d'associar Fòrum i Usuari associarem Fòrum i Moderador, ja que sabem que només el subconjunt d'usuaris que són moderadors estan capacitats per a moderar un fòrum.

Associacions



Resum

Hem vist com una eina que es va crear com a mecanisme d'abstracció per als llenguatges de programació ens permet simplificar la descripció de qualsevol sistema del món real. Per tant, hem de veure l'orientació a objectes no solament com un paradigma de programació, sinó com un paradigma d'anàlisi que podem aplicar al món real.

L'orientació a objectes ens proporciona una manera d'abstraure les característiques importants de les entitats que formen el nostre domini, classificar-les de manera que en puguem maximitzar la reutilització i que puguem variar el nivell d'abstracció al qual treballem i, al mateix temps, establir relacions entre aquestes mitjançant associacions.

Hem vist com podem classificar els objectes en classes i fer abstracció dels aspectes rellevants a l'hora de descriure les classes mitjançant les diferents responsabilitats que es traduiran en atributs, associacions i operacions.

També hem vist com l'encapsulament ens permet establir la visibilitat de cadascuna de les responsabilitats i ocultar, així, els detalls d'implementació a la resta de classes.

Finalment, l'herència ens permet establir jerarquies de classe de tal manera que puguem descriure els trets comuns a diverses classes en una classe més general, mentre que el polimorfisme ens permet veure un mateix objecte com d'una o altra classe de la jerarquia en funció del context.

Un model orientat a objectes del nostre domini ens facilitarà, més endavant, la creació de classes de programari que reflecteixin aquest domini, al mateix temps que ens pot ajudar a la creació d'altres models (com ara l'estructura de la base de dades) mantenint, al mateix temps, la possibilitat de fer-lo servir per a comunicar-nos amb les persones expertes en el domini sense necessitat que siguin expertes en programació orientada a objectes.

Activitats

1. A l'exemple del subapartat 2.3.1 hem dit que el telèfon és un atribut de tipus String. Per què no l'hem indicat com de tipus numèric?

2. Estem fent l'anàlisi orientada a objectes per a un sistema de comerç electrònic. L'empresa vendrà una sèrie de productes dels quals tindrem un nom, la marca comercial, una descripció, una fotografia i el preu. Alguns productes són electrònics, en el sentit que són documents binaris que es poden descarregar i no objectes físics que cal enviar; per a aquests, també volem tenir-ne el fitxer i el format, que pot ser música, llibre o pel·lícula. Cada producte podrà pertànyer a una o més categories, cadascuna de les quals tindrà un nom i una descripció.

Els usuaris tindran un nom d'usuari, una contrasenya i una adreça i faran compres de les quals desarem la data i el conjunt de productes comprats. D'un mateix producte, en una compra, se'n podran comprar una o més unitats. No tindrem en compte el pagament.

- Feu una llista de les classes que podem trobar en aquest domini.
- Per a cada classe, indiqueu els atributs que té, el tipus de cada atribut i quin és el seu nombre possible de valors.
- Identifiqueu les associacions entre classes. Per a cada una indiqueu els noms dels rols i, si n'hi ha, els atributs que tingui la classe associativa.
- Identifiqueu les relacions d'herència entre classes i quines classes creieu que són abstractes.

3. Tenim un sistema orientat a objectes amb les classes *Vehicle*, *Cotxe*, *Moto* i *Conductor*. La classe *Vehicle*, que és abstracta, té l'atribut públic *matrícula*, l'atribut protegit *marca* i l'atribut privat *model*. *Cotxe* i *Moto* són subclasses de la classe *Vehicle*.

a) Quins atributs de *Vehicle* són visibles des de la classe *Vehicle*?

b) I des de la classe *Cotxe*?

c) I des de *Persona*?

4. Tenim un sistema orientat a objectes amb les classes *Be*, *BeImmoble* (que és subclasse de *Be*) i *Contribuent*. Entre *Contribuent* i *Be* hi ha una associació, de tal manera que un contribuent té qualsevol nombre de béns, cadascun dels quals pertany a un o més contribuents. *Be* té l'atribut *valor*, mentre que *BeImmoble* té un atribut *referenciaCatastral*. *Be* té una operació *calcularImpostos* que calcula d'impost per aplicar als contribuents que tenen el bé associat. *BeImmoble* redefineix l'operació *calcularImpostos* i, a més, té una operació pròpia anomenada *hipotecar*, amb diversos paràmetres.

a) Pot passar que un contribuent tingui associat un bé immoble?

b) Un contribuent podria demanar l'operació *hipotecar* sobre tots els seus béns associats?

c) Segons aquest esquema, els béns immobles tenen valor?

d) Els béns tenen referència catastral?

5. Suposeu que tenim el mateix sistema orientat a objectes de l'exercici anterior. Ara, però, sabem que l'atribut *valor* de la classe *Be* és privat. I ara, els béns immobles tenen valor? Des de la classe *BeImmoble* es pot accedir a l'atribut *valor*?

Exercicis d'autoavaluació

1. Quina és la diferència fonamental entre els llenguatges de programació procedimentals i els orientats a objectes pel que fa a l'abstracció?

2. Quina relació hi ha entre l'anàlisi orientada a objectes i la programació orientada a objectes?

3. En orientació a objectes, amb quin criteri agrupa objectes el mecanisme de classificació?

4. Quins tipus de responsabilitats poden tenir els objectes?

5. Quin mecanisme de l'orientació a objectes ens permet descartar trets característics dels objectes que no siguin rellevants al problema que volem resoldre?

6. Quin element de la classe ens indica la informació que coneixem de les seves instàncies?

7. Quina informació coneixem dels atributs d'una classe, a més del nom?

8. Què vol dir que la relació d'associació sigui recíproca?

9. Pot existir una associació d'una classe amb si mateixa? En tal cas, com distingim una instància d'una altra entre dues que estan relacionades?

10. En orientació a objectes, com modelitzem les tasques que poden portar a terme els objectes?

11. Quina relació hi ha entre operacions, mètodes, interfície i implementació?

12. Quina altra metàfora es fa servir en orientació a objectes per a fer referència a la invocació o crida d'una operació?

13. Quina relació hi ha entre l'encapsulament i l'ocultació de la informació?

14. Quines visibilitats podem definir per a atributs i operacions en un sistema orientat a objectes?

15. Quin mecanisme de classificació ens permet simplificar la descripció de les classes?

16. Com podem identificar relacions d'herència en domini?

17. Quin mecanisme ens permet treballar amb un objecte en diversos nivells d'abstracció segons les nostres necessitats?

18. Què és una classe abstracta?

19. Una classe que no és abstracta pot tenir operacions abstractes? Per què?

20. Una classe abstracta pot tenir operacions que no ho siguin? Per què?

21. Quina relació hi pot haver entre el concepte d'associació i el de polimorfisme?

22. Com podem solucionar, en orientació a objectes, el cas en què una associació té informació addicional que no depèn d'una de les instàncies relacionades, sinó de totes?

Solucionari

Activitats

1. Els telèfons són cadenes de caràcters que, aparentment, només poden contenir nombres, però no són nombres en el sentit que no els percebem com a tals i no els fem servir com a tals en cap moment. Així, per exemple, alguns trets que els distingeixen dels nombres són:

- No és el mateix el telèfon 003 que el 3, tot i que sí que són el mateix nombre.
- Els telèfons no admeten cap de les operacions habituals dels nombres. Mai no sumem, restem, multipliquem, dividim, etc. telèfons, tot i que sí que ho fem amb els nombres.
- Alguns caràcters no numèrics poden formar part d'un número de telèfon. Com a mínim el caràcter + que podem fer servir per a senyalitzar el prefix internacional d'un número de telèfon, i que realment es marca en marcar el número.

2. Classes que podem trobar i atributs:

- Producte: nom:String, marcaComercial:String, descripcio:String, fotografia:Imatge, preu:Import
- Producte electrònic: fitxer:Fitxer, format:Format
- Categoria: nom:String, descripcio:String
- Usuari: nomUsuari:String, contrassenya:String, adreça:Adreça
- Compra: data:Data

Associacions:

- PertanyA: cada producte (amb nom de rol *productes*) pertany a una o més categories (amb nom de rol categories).
- Fa: cada usuari (nom de rol *usuari*) fa un seguit de compres (nom de rol *compres*).
- Inclou: una compra (nom de rol *compra*) inclou un seguit de productes (nom de rol *productes*). Aquesta és una classe associativa amb l'atribut *numUnitats:nat*.

Hi ha una relació d'herència entre *Producte* i *Producte electrònic*, de tal manera que *Producte electrònic* és una subclasse de *Producte*. Producte, però, no és una classe abstracta, ja que hi ha productes que no són electrònics.

3.

a) Tots
b) Matrícula i marca, ja que *marca* és protegit i és visible des de les subclasses però *model* és privat i no és visible des de cap altra classe que no sigui *Vehicle*.
c) *Matrícula*, ja que *marca* i *model* no són públics.

4.

a) Sí. Un contribuent pot tenir associat qualsevol nombre de béns i els béns immobles també són béns.
b) No, ja que els béns que no són immobles no es poden hipotecar; és a dir, l'operació només està definida per a la classe *BeImmoble* i, per tant, només es pot invocar sobre instàncies d'aquesta classe.
c) Sí. Tot bé immoble és un bé i, per tant, com tots els béns, té un valor. Vist des del punt de vista de l'herència, la classe *BeImmoble* hereta l'atribut valor de la classe *Be*.
d) En general, no. Només aquells béns que siguin immobles tindran referència cadastral. Però tots els altres no en tindran.

5. La visibilitat d'un atribut no en canvia el comportament pel que fa a l'herència. Per tant, si *valor* és un atribut privat, tots els béns continuaran tenint valor, incloent-hi els béns immobles. Ara bé, com que l'atribut *valor* és ara privat, encara que els béns immobles també en tinguin, no hi podem accedir des de la classe *BeImmoble*.

Exercicis d'autoavaluació

1. Els llenguatges de programació procedimentals fan una abstracció del que la màquina pot entendre, mentre que els llenguatges de programació orientats a objectes fan una abstracció d'allò que les persones volen comunicar. (Vegeu el subapartat 1.2.)

2. L'anàlisi orientada a objectes consisteix a aplicar els conceptes i idees dels llenguatges de programació orientats a objectes a l'anàlisi de sistemes. Per tant, l'anàlisi orientada a objectes fa servir el mateix paradigma, l'orientació a objectes, que la programació orientada a objectes. (Vegeu el subapartat 1.3.)

3. El criteri pel qual la classificació agrupa objectes en classes és el conjunt de responsabilitats en comú que tenen aquests objectes. Així, tots els objectes d'una mateixa classe tenen unes mateixes responsabilitats. (Vegeu el subapartat 2.1.)

4. Un objecte pot tenir responsabilitats de dades (coses que ha de saber), associacions (altres objectes que coneix) i comportament (tasques que se li poden demanar). (Vegeu el subapartat 2.1.)

5. L'abstracció. És el mecanisme mitjançant el qual decidim quines de les possibles responsabilitats associades a una classe formaran part de la seva definició. (Vegeu el subapartat 2.2.)

6. Els atributs. (Vegeu el subapartat 2.3.1.)

7. Els atributs solen tenir un tipus, que ens ajuda a entendre el tipus d'informació que s'hi emmagatzema, un àmbit (que pot ser d'instància o de classe) i un nombre de valors (pot ser o no opcional i pot ser o no multivaluat). (Vegeu el subapartat 2.3.1.)

8. Vol dir que si una instància a^1 d'una classe A està associada amb una instància b^1 de la classe B, la instància b^1 estarà també associada a la instància a^1. (Vegeu el subapartat 2.3.2.)

9. Si, i s'anomena *associació recursiva*. Per a distingir el rol que té cada instància de l'associació fem servir els rols d'associació. (Vegeu l'apartat 2.3.2.)

10. Mitjançant operacions. (Vegeu el subapartat 2.3.3.)

11. Les operacions indiquen quines tasques pot dur a terme un objecte i formen la interfície de l'objecte. Cada mètode és el conjunt d'instruccions que indica com un objecte du a terme una tasca (una operació) i forma la implementació de l'objecte. (Vegeu el subapartat 2.3.3.)

12. També es fa servir la metàfora de l'enviament d'un missatge. Quan un objecte invoca o crida una operació d'un altre objecte també podem dir que li envia un missatge. (Vegeu el subapartat 2.3.3.)

13. L'encapsulament és el mecanisme de l'orientació a objectes que ens permet ocultar informació.

14. Visibilitat pública (qualsevol objecte hi té accés), privada (només els objectes de la mateixa classe), protegida (només els objectes de la mateixa classe o d'alguna subclasse) i de paquet (només els objectes del mateix grup o "paquet"). (Vegeu el subapartat 3.2.)

15. L'herència. Ens permet simplificar la descripció d'una classe identificant responsabilitats comunes entre les instàncies de diferents classes i evitant haver-les de descriure en més d'una classe. (Vegeu l'apartat 4.1.)

16. Mitjançant la generalització (trobem classes més generals a partir de les que ja tenim) o l'especialització (trobem classes més específiques a partir de les que ja tenim). (Vegeu el subapartat 4.1.1.)

17. El polimorfisme. És la capacitat de treballar amb un objecte amb formes diferents (interfícies diferents) segons les nostres necessitats. (Vegeu el subapartat 4.2.)

18. Una classe abstracta és una classe que no podem instanciar directament; és a dir, només podem crear instàncies d'alguna de les seves subclasses. (Vegeu el subapartat 4.2.1.)

19. Una classe no abstacta no pot tenir operacions abstractes, perquè, per a aquestes, el comportament (el mètode) s'ha de definir en les subclasses. Per tant, els objectes de la classe no tindran cap comportament associat, tret que pertanyin a una subclasse. Per això, per força, haurien de pertànyer a una subclasse; és a dir, la classe hauria de ser abstracta. (Vegeu el subapartat 4.2.1.)

20. Una classe abstracta pot tenir operacions que no ho siguin. Senzillament, les seves subclasses heretaran les operacions amb el seu comportament i el podran redefinir o deixar-lo tal qual. (Vegeu el subapartat 4.2.1.)

21. Podem tenir associacions polimòrfiques: Una associació polimòrfica és una associació amb una classe que té subclasses. En aquest cas, un objecte pot tenir associats, per una mateixa associació, objectes de la classe associada i de les seves subclasses. Per tant, tindrà associats objectes de diverses classes (amb una superclasse en comú). (Vegeu el subapartat 4.2.2.)

22. Fem servir una classe associativa, és a dir, una classe les instàncies de la qual són instàncies de l'associació. (Vegeu l'apartat 2.3.2.)

Glossari

abstracció *f* L'acte de reunir les qualitats essencials o generals de coses similars. També, les característiques essencials provinents d'una cosa.

abstracta (classe) *f* Classe que no es pot instanciar directament, sinó que només es poden instanciar les subclasses no abstractes que tingui.

abstracta (operació) *f* Operació que no té associat un mètode; és a dir, per a la qual no s'ha definit un comportament.

àmbit (d'un atribut) *m* Característica que distingeix si un atribut emmagatzema un valor diferent per a cada instància (àmbit d'instància) o bé un mateix valor per a totes les instàncies d'una classe (àmbit de classe).

anàlisi orientada a objectes *f* Aplicació dels conceptes i idees dels llenguatges orientats a objectes a l'anàlisi de sistemes.

associació *f* Relació estructural entre dues (o més) classes que ens indica quins objectes de la segona classe coneixen els objectes de la primera. Una associació, per tant, constitueix una responsabilitat de cada una de les classes implicades.

associació polimòrfica *f* Vegeu **polimòrfica (associació)**

associació recursiva *f* Associació que connecta una classe amb ella mateixa.

atribut *m* Element estructural d'una classe que indica quina informació tenen les instàncies d'aquella classe; és una de les responsabilitats, per tant, de la classe.

caixa blanca *f* Tipus de descripció d'un objecte en què es descriu, a més de qualsevol aspecte extern de l'objecte, la seva estructura i funcionament interns.

caixa negra *f* Tipus de descripció d'un objecte en què es descriu el seu funcionament des d'un punt de vista extern, sense explicar-ne l'estructura ni el funcionament interns.

classe abstracta *f* Vegeu **abstracta (classe)**

classe associativa *f* Classe de la qual les instàncies són instàncies d'una associació; és a dir, cada instància d'una classe associativa representa una instància d'una associació en el sentit que està formada pel parell (en el cas d'associacions binaries) d'instàncies associades, a més de tenir els atributs, associacions i operacions propis.

classe (d'objectes) *f* Descriptor d'un conjunt d'objectes que comparteixen els mateixos atributs, associacions, operacions i mètodes.

classificació *f* Mecanisme d'identificar classes d'objectes com a descriptores de conjunts d'objectes amb les mateixes responsabilitats.

codi màquina *m* Codi format pel conjunt d'instruccions que els processadors (en general, el maquinari) poden entendre i executar.

cridar *f* Vegeu **invocar**.

de paquet (visibilitat) Visibilitat d'un atribut, associació o operació tal que només és visible pels objectes del mateix grup o paquet.

encapsulament *m* Mecanisme de l'orientació a objectes que ens permet ocultar informació definint quins atributs, operacions i associacions d'un objecte són visibles i quins resten ocults.

especialització *f* Procés d'identificació d'una subclasse d'una classe existent que serveix per a descriure les responsabilitats comunes a un subconjunt de les instàncies de la classe original, però no a totes.

generalització *f* Procés d'identificació d'una superclasse de dues o més classes ja existents que serveix per a descriure les responsabilitats comunes a les classes ja identificades inicialment.

herència *f* Característica de l'orientació a objectes mitjançant la qual podem identificar una relació entre una superclasse i una subclasse tal que totes les instàncies de la subclasse ho són també de la superclasse. Diem que la subclasse hereta la definició de la superclasse,

de tal manera que per a definir la subclasse només ens cal indicar aquelles característiques que li són específiques que no són presents a la superclasse.

implementació (d'una classe) *f* Comportament intern de les instàncies d'una classe que queda definit pel conjunt de mètodes de la classe. La implementació queda oculta darrere la interfície.

instància *f* Vegeu **objecte**.

interfície *f* Conjunt d'operacions ofertes per les instàncies d'una classe i que, per tant, són invocables per altres objectes. La interfície no inclou, però, la implementació, que està formada pels mètodes que implementen aquestes operacions.

invocar *v* Demanar a un objecte l'execució del seu comportament per a una determinada operació.

llenguatge assemblador *m* Llenguatge de programació en què les instruccions es corresponen pràcticament una a una amb les operacions que pot entendre el processador. Es tradueix a codi màquina per a ser executat.

llenguatge de programació *m* Llenguatge artificial que està dissenyat per a expressar els còmputs que volem que dugui a terme un ordinador.

llenguatge de programació d'alt nivell *m* Llenguatge de programació més ric que el llenguatge assemblador, ja que permet expressar les instruccions amb un nivell d'abstracció més alt.

llenguatge de programació orientat a objectes *m* Llenguatge de programació d'alt nivell que eleva el nivell d'abstracció creant una abstracció del problema que es vol resoldre fent servir el paradigma de l'orientació a objectes. Vegeu **orientació a objectes**.

llenguatge natural *m* Llenguatge que fem servir les persones per a comunicar-nos entre nosaltres (com ara el català o l'anglès), per oposició als llenguatges que fem servir per a comunicar-nos amb els ordinadors (com ara els llenguatges de programació).

llenguatge procedimental *m* Llenguatge de programació d'alt nivell que eleva el nivell d'abstracció creant una abstracció de la màquina sobre la qual s'executen. Permet fer servir un joc d'instruccions més abstractes que les que el processador és directament capaç d'executar.

llenguatge unificat de modelització *m* Vegeu ***universal modeling language***

mètode *m* Conjunt d'instruccions que, per a una operació, en una classe, indica com es comportaran els objectes d'aquella classe. El conjunt dels mètodes de la classe en forma la implementació.

missatge (orientació a objectes) *m* Metàfora que representa una crida o invocació d'una operació d'un objecte. Sota aquesta metàfora diem que l'objecte que fa la crida (el que demana l'execució de l'operació) envia un missatge a l'objecte que rep la crida (el que ha d'executar el seu comportament per a aquella operació).

multivaluat (atribut) *m* Atribut que, per a una determinada instància, pot tenir més d'un valor en lloc d'haver-ne de tenir només un. Antònim d'*univaluat*.

objecte *m* Cada un dels elements autònoms que forma un sistema orientat a objectes

obligatori (atribut) *m* Atribut tal que totes les instàncies de la classe hi han de tenir valor. Antònim d'*opcional*.

ocultació d'informació *f* Capacitat de fer que cada element d'un sistema contingui certa informació que s'amaga a la resta d'elements, mostrant-los una visió simplificada. En el cas de les classes d'un sistema orientat a objectes, l'encapsulament permet indicar quins elements són visibles i ocultar la resta.

opcional (atribut) *m* Atribut que, per a una determinada instància, pot no tenir valor. Si un atribut no és opcional, aleshores ha de tenir valor per a totes les instàncies de la classe. Antònim d'*obligatori*.

operació *f* Especificació d'una tasca que poden dur a terme totes les instàncies d'una classe. L'operació es pot demanar mitjançant la crida o invocació i el comportament de la instància que rebi la crida es defineix en un mètode. És, per tant, una de les responsabilitats de la classe.

operació abstracta *f* Vegeu **abstracta (operació)**

operació polimòrfica *f* Vegeu **polimòrfica (operació)**

orientació a objectes *f* Paradigma consistent a descompondre un sistema complex en un conjunt d'elements autònoms (els objectes) que es comuniquen els uns amb els altres per tal de dur a terme les seves tasques.

paquet *m* Agrupació d'elements en un sistema orientat a objectes per a facilitar-ne l'organització. Un paquet pot contenir classes i altres paquets i pot resultar útil per a definir la visibilitat de paquet dels atributs, associacions i operacions de les classes que conté.

polimòrfica (associació) *f* Associació en què una (o més) de les classes que hi participen té subclasses, de tal manera que un objecte pot tenir associats, per la mateixa associació, objectes pertanyents a diferents classes (sempre que pertanyin a la jerarquia d'herència de la classe associada).

polimòrfica (operació) *f* Operació que té un mètode associat en una classe però que té un mètode diferent per a alguna de les subclasses de la classe. D'aquesta manera, les diferents instàncies de la classe, en executar l'operació, es comportaran de manera diferent en funció de si són instàncies només de la classe o de quina subclasse siguin instàncies.

polimorfisme *m* Capacitat d'un objecte de presentar-se amb formes diferents (interfícies diferents) segons el context. Així, un objecte que pertanyi a una classe qualsevol es pot veure com a pertanyent a aquella classe, a la superclasse d'aquella, a la superclasse de la superclasse d'aquella, etc.

protegida (visibilitat) *f* Visibilitat d'un atribut, associació o operació tal que només és visible per la classe en què està definida i les seves subclasses.

privada (visibilitat) *f* Visibilitat d'un atribut, associació o operació tal que només és visible per la mateixa classe en què està definida.

pública (visibilitat) *f* Visibilitat d'un atribut, associació o operació tal que és visible per tots els objectes del sistema.

redefinir (una operació) *v* Definir, per a una operació d'una subclasse, un mètode (un comportament) diferent del mètode que tenia associada aquella operació a la superclasse. En redefinir una operació en una subclasse la convertim en una operació polimòrfica.

responsabilitat (d'un objecte) *f* Característiques o obligacions d'una classe, que poden ser informació que les instàncies han de conèixer (atributs), altres instàncies que han de conèixer (associacions) o tasques que han de poder dur a terme (operacions).

rol *m* Extrem d'una associació al qual s'assigna un nom per a indicar el seu propòsit; és a dir, per a indicar quin és el paper que hi tenen les instàncies quan s'associen.

subclasse *f* En una relació d'herència entre dues classes, la classe que hereta les responsabilitats de l'altra classe; és, per tant, la classe més específica.

superclasse *f* En una relació d'herència entre dues classes, la classe de la qual s'hereten les responsabilitats; és, per tant, la classe més genèrica.

tipus (d'un atribut) *m* Descripció del conjunt de valors que pot prendre un atribut. Alguns exemples de tipus d'atributs poden ser el tipus data (que defineix un conjunt de valors que inclou 1/3/2011 o 31/3/1414), enter (com ara 14 o 356) o caràcter (com ara "a" o "@").

unified modeling language *m* Llenguatge de modelització de propòsit general estandarditzat per l'OMG i utilitzat en el camp de l'enginyeria del programari.
Sigla **UML**

univaluat (atribut) *m* Atribut tal que cap instància de la classe no pot tenir més d'un valor en aquell atribut. Antònim de *multivaluat*.

visibilitat *f* Propietat d'un atribut, associació o operació que defineix quins objectes el poden veure o veure-la. La visibilitat pot ser pública, privada, protegida o de paquet.

Bibliografia

La bibliografia sobre orientació a objectes està molt relacionada amb el món de la programació.

Meyer, Bertrand (2002). *Construcción de software orientado a objetos*. Madrid: Prentice Hall.

Aquest llibre és la referència acadèmica en orientació a objectes i, per tant, és molt adequat per a l'estudi en profunditat dels diferents conceptes.

Bibliografia complementària

Moltes guies i llibres sobre programació en un llenguatge orientat a objectes inclouen una secció introductòria sobre la programació a objectes. A continuació us en recomanem dos:

Eckel, Bruce (2006). *Thining in Java*. Prentice Hall.

Object Oriented Programming with Objective C (Apple Corp.) http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/OOP_ObjC/OOP_ObjC.pdf

Referències bibliogràfiques

Parnas, David L. (1972). "On the Criteria to Be Used in Decomposing Systems Into Modules". *Communications of the ACM*.