Introducció al paradigma de la programació orientada a objectes

PID_00269654

David García Solórzano

Temps mínim de dedicació recomanat: 2 hores

Universitat Oberta de Catalunya

David García Solórzano

Graduat Superior en Enginyeria en Multimèdia i Enginyer en Informàtica per la Universitat Ramon Llull des de 2007 i 2008, respectivament. És també doctor per la Universitat Oberta de Catalunya des de 2013, en la qual va presentar una tesi doctoral relacionada amb l'àmbit de l'*e-learning*. Des de 2008 és professor de la Universitat Oberta de Catalunya en els Estudis d'Informàtica, Multimèdia i Telecomunicació.

L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: David García Solórzano (2020)

Primera edició: febrer de 2020

Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Índex

Introducció

Aquest mòdul vol situar-nos en el punt de partida des del qual començaran aquests materials docents, que no és un altre que el paradigma de la programació estructurada que ja hauríem de conèixer (i dominar). Una vegada que estiguem situats, descriurem els avantatges del paradigma de la programació orientada a objectes (POO) respecte al paradigma de la programació estructurada. A més, explicarem els elements bàsics de la POO i veurem com s'ha de pensar a l'hora de dissenyar una solució (és a dir, un programa) seguint el paradigma de la POO.

Aquest mòdul és introductori i, per tant, molts dels conceptes que s'hi comenten es veuran amb més detall en mòduls futurs.

Aquests materials didàctics no són:

- d'algorítmica (donem per fet que tens experiència prèvia en programació estructurada i que coneixes perfectament què és un condicional, un `int`, un `boolean`, un bucle, un *array*, una funció, un pas per valor i per referència, un punter, una pila, una cua, una llista, etc.),
- principalment de programació, com podrien ser els materials d'una assignatura inicial,
- únicament de disseny de programes, com podrien ser els materials d'una assignatura d'enginyeria de *software*,
- de Java, ni C#, ni C++, ni Python, ni PHP, ni Scala... No són de cap llenguatge de programació orientat a objectes i ho són de tots.

Aquests materials són un pas més enllà, ja que se centren en l'ensenyament d'un paradigma, concretament del paradigma de la programació orientada a objectes (POO). Per tant, aquests materials:

- combinen coneixements de disseny i programació per donar una solució *software* a problemes reals,
- expliquen conceptes generals de la POO que serveixen per a «qualsevol» llenguatge de programació que suporti aquest paradigma (veurem que hi ha característiques que de vegades no són suportades per tots els llenguatges o, que si les suporten, ho fan amb matisos),

- intenten, tant com sigui possible, allunyar-se de la codificació per aprofundir en els conceptes, en les idees i, en definitiva, en la filosofia que hi ha darrere de cada aspecte de la POO,

- si mostren codi, utilitzen un pseudocodi que simplifica la verbositat dels llenguatges de programació per ressaltar els conceptes més que la sintaxi a emprar. No obstant això, hi ha aspectes la codificació dels quals sí que es mostra en diferents llenguatges reals, predominant els exemples en Java, C# i C++.

Aclarit què són aquests materials i què no, et formulem una pregunta: Estàs disposat a treballar amb aquests materials gairebé diàriament i durant mesos? Esperem que la teva resposta hagi estat «sí», sense condicions ni però. Nosaltres ens comprometem a explicar-te les coses de la manera més senzilla i amena de què siguem capaços (això sí: tu també hi hauràs de posar de la teva part!). Per aconseguir-ho, sobretot al principi, seguirem la família POO –ja te la presentarem– en la seva vida quotidiana per poder fer analogies del món real amb alguns conceptes de la POO.

D'altra banda, et volem recordar que aquests materials són principalment teòrics. I la pràctica? L'única manera de traslladar els coneixements teòrics a *software* real és utilitzant un llenguatge de programació concret. Per aquest motiu, en paral·lel a aquests materials didàctics, et facilitem una guia que explica el llenguatge de programació escollit durant el semestre i que detalla com els conceptes vistos en aquests materials didàctics es posen en pràctica.

Finalment, esperem que gaudeixis de l'aprenentatge d'un paradigma tan potent com és la programació orientada a objectes. Sabem que és un repte que exigeix molt d'esforç per part teva, però t'assegurem que la recompensa final ho mereix.

Objectius

L'objectiu principal d'aquest mòdul és contextualitzar el material docent, concretament:

1. Ser conscient del punt de partida d'aquests materials: el paradigma de la programació estructurada.

2. Conèixer el concepte de *paradigma de programació* fent èmfasi en la programació estructurada –ja coneguda per l'estudiant– i la seva evolució cap a la programació orientada a objectes (POO) –paradigma que s'estudiarà en aquests materials.

3. Comprendre de manera general els beneficis que ens aporta el paradigma de la programació orientada a objectes.

4. Entendre els elements bàsics de la POO, com també el raonament, en línies generals, que se segueix en dissenyar un programa mitjançant el paradigma POO.

5. Ser conscient de les dificultats que un programador novell afronta en intentar aprendre el paradigma de la programació orientada a objectes, i l'actitud (i esforç) que aquest exigeix.

1. Paradigma de la programació estructurada: d'on venim?

Si estàs llegint aquestes línies és perquè tens nocions bàsiques de programació. Si és així, llavors segur que ets conscient que un programa s'ha de dissenyar acuradament (és a dir, pensar amb deteniment) abans de codificar-lo. Això és més cert com més complexa és la tasca que ha de dur a terme el programa.

Per a garantir que els programes complexos funcionen –és a dir, fan el que se n'espera–, són fiables i estan ben codificats, a la fi dels anys cinquanta va aparèixer una nova disciplina anomenada **enginyeria del *software***. Aquesta disciplina proposa mètodes i teories que serveixen tant per a analitzar problemes complexos com per a dissenyar programes que resolguin aquests problemes. Cada conjunt de mètodes i teories que determinen una manera específica d'analitzar i resoldre un problema s'anomena *enfocament de programació* o, més formalment, **paradigma de programació**.

Definició de paradigma de programació

Un paradigma de programació és una col·lecció de conceptes, mètodes i teories que guien el procés de construcció d'un programa determinant-ne l'estructura.

Així, doncs, un paradigma de programació determina la manera en què pensem i formulem els programes.

En els anys setanta, el paradigma estrella era l'anomenat **programació estructurada**. Corrado Böhm i Giuseppe Jacopini van formalitzar aquest paradigma el 1966. La programació estructurada es basa a seguir un enfocament descendent (*top-down*) per a dissenyar un programa que resolgui un problema complex. Per a això, es descompon el problema o programa inicial en diversos problemes o programes més petits i senzills anomenats *subproblemes*. Cada subproblema es treballa individualment i es considera un nou problema que es pot descompondre, al seu torn, en problemes més petits i així successivament. Finalment, s'arriba a problemes tan senzills que es poden resoldre directament sense necessitat de descompondre'ls més.

Aquest enfocament *top-down* implica tenir diferents nivells de descomposició que van del general –el problema complex inicial– al particular –el subproblema de més baix nivell de descomposició.

En aquest punt, si diem que el problema complex inicial és el programa, llavors també podem dir que un subproblema és un subprograma. Els subprogrames del primer nivell de descomposició se solen anomenar **mòduls**. Els nivells de descomposició posteriors als mòduls estan formats per **funcions** i **accions** – és a dir, funcions que no retornen res.

Definició de mòdul

Un mòdul és una col·lecció que agrupa constants, variables, funcions i accions relacionades entre elles.

El nombre de mòduls, i també de funcions i d'accions, que té un programa depèn de cada programador. Cada mòdul sol encarregar-se d'una funcionalitat del programa i aquests poden interactuar entre ells. Així, doncs, si el nostre programa és un videojoc de bàsquet, el mòdul de física dirà al mòdul de gràfics on ha de dibuixar la pilota.

El paradigma de la programació estructurada té els avantatges següents respecte a no seguir cap paradigma:

- **Millora la llegibilitat del codi.** Tenir el codi organitzat en mòduls que segueixen una certa lògica i en funcions de poques línies que fan una tasca concreta és millor que tenir un únic mòdul amb milers de línies de codi. Així mateix, els programes queden millor documentats internament.

- **Facilita el treball en grup.** Diversos programadors poden treballar en el mateix programa simultàniament encarregant-se cadascun d'un mòdul concret.

- **Augment de la productivitat.** Com que el codi està organitzat en mòduls, es pot ajornar el desenvolupament d'algunes tasques, destinar més programadors a un mòdul concret segons les necessitats del moment i augmentar així la productivitat de l'equip.

- **Manteniment més senzill.** Atès que la lògica del programa és més visible gràcies a la seva pròpia estructuració, els errors es poden localitzar i corregir més fàcilment. Així, doncs, si el videojoc de bàsquet esmentat anteriorment falla a l'hora de pintar la pilota, sabem que l'error està en alguna funció o acció del mòdul de gràfics. A més, si es requereix modificar algun segment de codi (p. ex. un mòdul), es pot fer sovint sense tocar la resta del programa.

- **Codificació més simple.** La pròpia organització o estructura del programa implica que la codificació sigui més fàcil per al programador, ja que a cada moment s'ha de centrar a resoldre programes o problemes senzills i petits.

És important tenir present que el grau d'èxit que podem tenir en seguir el paradigma de la programació estructurada és directament proporcional a la capacitat d'**abstracció** que siguem capaços d'aplicar al problema complex al qual ens enfrontem. La nostra capacitat d'abstracció ens permet reconèixer els diferents subproblemes del problema complex inicial.

Tot el que hem explicat fins ara t'hauria de ser familiar, ja que a dia d'avui ja hauries d'haver implementat programes seguint el paradigma de la programació estructurada utilitzant un llenguatge afí a aquest paradigma, com és el C. Si no recordes si has usat o no el paradigma de la programació estructurada, contesta a aquestes preguntes:

- **He usat mòduls?** Sí. Quan escrivies un programa (p. ex., en C, PHP, JavaScript, etc.) organitzaves el codi en fitxers (amb extensió `.c`, `.php`, `.js`, etc.) que contenien variables i funcions.

- **He seguit un enfocament descendent (*top-down*)?** Sí. A més de mòduls, quan escrivies una funció `F` i aquesta feia moltes tasques, la subdividies en funcions més senzilles que cridaves des de la funció `F`.

- **He aplicat abstracció?** Sí. En cada nivell de descomposició eres capaç de considerar solament els aspectes necessaris per a analitzar el problema i prendre decisions. Per exemple, decidies sobre qüestions com: en quants mòduls calia dividir el programa, què feia cada mòdul, quines funcions implementaves en cada mòdul, quina capçalera o signatura tenia cada funció, quines variables necessitaves i de quin tipus eren, etc.

Si has contestat «sí» a les tres preguntes anteriors, llavors estàs preparat per a continuar llegint. En cas contrari, seria recomanable que repassessis tot el que has fet de programació fins avui.

2. Paradigma de la programació orientada a objectes: on anem?

Si bé la programació estructurada és útil, la complexitat que han anat adquirint els programes al llarg dels anys ha obligat a pensar nous paradigmes. Així és com en els anys vuitanta va irrompre amb força un nou paradigma anomenat **programació orientada a objectes (POO)** –*object-oriented programming* (OOP).

Per a entendre el perquè de l'èxit de la POO cal pensar en els principals problemes que té la programació estructurada i com la POO els soluciona.

2.1. Reutilització de codi

Vegem dos exemples quotidians amb la família POO relacionats amb la reutilització:

Exemple 1 (Reutilització) – La CPU espatllada

Al David, pare de la família POO, se li ha espatllat la CPU de l'ordinador. Què fa? Va a una botiga que ven components de maquinari, compra una CPU nova compatible amb la resta de components de l'ordinador, se'n va a casa i canvia la CPU espatllada per la nova. Encén l'ordinador i funciona!

Exemple 2 (Reutilització) – La cadira del cotxe

L'Elena, mare de la família POO, s'ha emportat el cotxe sense recordar-se que el David ha de portar la seva filla Marina a classes de natació a 10 km de casa i no hi ha un transport públic que els deixi a prop. Per sort, el David té una cadira del grup 1-2 al traster. Què fa? Truca a l'avi Manel, que va a buscar-los amb el seu cotxe; instal·len la cadira per a nens als seients posteriors del vehicle i, solucionat! La Marina no es perdrà la seva classe de natació.

Ara pensa, podem reutilitzar el codi o el *software* que hem escrit en el passat en un altre programa tan fàcilment com es reutilitza el maquinari? La resposta és «no». La programació estructurada se centra en els procediments i és la funció la unitat bàsica. Les funcions són poc reutilitzables, ja que és molt difícil copiar una funció d'un programa a un altre i que funcioni a la primera. Per exemple, moltes funcions criden altres funcions del mòdul en el qual es troben –fins i tot d'altres mòduls– i que també s'haurien de cridar (i copiar) al nou programa. Algunes funcions fins i tot fan ús de variables globals –sí, ja sabem que aquest tipus de variables no són adequades, però són molt freqüents– i fallarien en la seva execució en el nou programa. Així, doncs, les funcions no són adequades per a la reutilització.

En canvi, **la POO afavoreix la reutilització de codi** en altres programes mitjançant l'ús de classes. Com veurem, una classe combina o encapsula en un mateix element estructures de dades (i.e. variables) i algorismes (i.e. funcions). Així, doncs, gràcies a la POO, no necessitem reescriure les mateixes funcions

una vegada i una altra en diferents programes, sinó que podem reutilitzar un codi –és a dir, una classe– que ja existeix –completament provat– en altres programes, de manera fàcil, ràpida i segura.

Com veurem en l'últim mòdul d'aquests materials, una de les maneres de facilitar la reutilització és mitjançant el mecanisme d'*herència*. L'herència permet bàsicament dues coses:

- Crear un mòdul –millor dit, una classe– a partir d'una altra ja existent i estendre així les funcionalitats de la classe reutilitzada sense afectar-la.

- Definir i utilitzar de manera clara classes funcionalment incompletes però que ajuden a definir la solució al problema.

2.2. Abstracció a alt nivell

Els llenguatges de programació vinculats al paradigma estructurat, com el C o el Pascal, obliguen el programador a pensar en termes propis de l'ordinador (p. ex., ús de la memòria, bucles, condicionals, etc.) més que en termes del problema que intenta resoldre. És a dir, el nivell d'abstracció al qual arriben és molt baix i es queda a nivell de sentències de programació. Quantes vegades hem pensat: què faig servir, un `if` o un `switch`? un `for` o un `while`? un `int` o un `float`? Com veiem, fins avui, els nostres programes han estat tan senzills que la majoria de les nostres decisions s'han centrat més en el codi en si mateix que en la funcionalitat del programa.

No obstant això, els llenguatges de programació orientats a objectes forcen el programador a centrar-se en el problema que cal resoldre, en comptes d'en el codi. Així, doncs, el disseny del programa o *software* guanya molt protagonisme, la qual cosa obliga el programador a pensar en un nivell d'abstracció més alt. El fet que el programador dediqui gran part del seu temps al disseny del *software* porta a un desenvolupament més productiu i de més qualitat.

Amb l'experiència veuràs que és molt més natural i en concordança amb la realitat pensar en un problema en termes de classes o objectes (com fa la POO) que pensar en funcions (com fa la programació estructurada). Així, doncs, en la POO, la manera de dissenyar un *software* canvia de *top-down* a *bottom-up*; de fet, es parteix d'entitats reals que hi ha en el problema que cal resoldre i són «traduïdes» a entitats *software* que interactuen entre elles. No és senzill fer aquest canvi en la nostra manera de pensar i dissenyar el *software*, però a poc a poc, amb la pràctica, la nostra ment s'hi va acostumant.

2.3. Seguretat

Molts mòduls de programes basats en el paradigma de la programació estructurada inclouen variables globals que poden ser usades per altres mòduls accedint-hi directament o mitjançant funcions proporcionades pel mòdul que les declara. Al començament dels vuitanta, es va popularitzar la segona manera d'accedir-hi, és a dir, els mòduls d'un programa no accedien directament a les variables d'un altre mòdul, sinó que ho feien mitjançant funcions proporcionades pel mòdul que contenia les variables que s'usarien. Això té una sèrie d'avantatges respecte a accedir directament com, per exemple, els següents:

- S'assegura la **consistència** de la variable. Per exemple, si una variable és un *array*, en usar una funció per a accedir-hi, serà aquesta funció la que comprovi si l'índex al qual es vol accedir (passat per paràmetres) és correcte o no. A més, es garanteix que tots els accessos a l'*array* fets mitjançant la funció es faran de la mateixa manera i es duran a terme les mateixes comprovacions.

- S'**eximeix de responsabilitats** el programador que utilitza el mòdul, ja que, en usar les funcions proporcionades, la responsabilitat recau en la persona o persones que han programat aquestes funcions.

- **Facilitat de manteniment**. Si es canvia el tipus de variable o hi ha algun error en una de les funcions proporcionades pel mòdul, solament cal canviar el codi del mòdul que la conté. D'aquesta manera, els canvis es veuen reflectits automàticament en els mòduls que la utilitzin.

Com podem apreciar, a poc a poc va sorgir la tendència d'anar ocultant informació sobre la implementació dels mòduls o, dit d'una altra manera, de fer d'un mòdul una caixa negra de la qual se sabés el mínim perquè un tercer la pogués usar. Aquest hàbit d'ocultar informació i de controlar-ne l'accés –especialment dades, és a dir, variables– està íntimament relacionat amb els conceptes d'**encapsulació** (una de les característiques principals de la POO) i **ocultació d'informació (o nivells o modificadors d'accés)**. Així, doncs, el programador coneix una interfície ben definida que indica quines funcions conté un mòdul (en POO serà una classe) i que, per tant, pot utilitzar o cridar. Vegem-ne un exemple amb el paradigma de la programació estructurada:

Exemple 3 (Encapsulació) – Què preferiu...?

Imaginem que necessitem usar un mòdul fet per una tercera persona en el nostre programa. Pensem, en aquest context, què és el més important per a nosaltres: saber com està implementat aquest mòdul per dins o saber què és el que podem fer-ne i com podem utilitzar-lo? Efectivament, el segon. A qualsevol programador que usa codi d'un tercer, el que li interessa saber és quines funcions pot usar, què fan, com s'usen (és a dir, com es criden) i què retornen. Per contra, no li interessa gens ni mica si el programador que va fer el mòdul, en escriure una funció, va utilitzar una sèrie de sentències `if` o va preferir un `switch`, o si va usar un `while` en comptes d'un `for`, etc.

Així, doncs, l'ideal per a nosaltres és incloure el nou mòdul en el nostre programa, que ens diguin que hi ha una funció anomenada `parellOSenar` que rep un nombre i imprimeix per pantalla si el nombre és parell o senar. Ens és igual com aquesta funció aconsegueix

saber si el nombre passat com a argument és parell o senar, el que ens importa és que en algun lloc del nostre codi farem `parellOSenar(2)` i per pantalla apareixerà "`2 és parell`".

Atesos els avantatges, els programadors tenien cada vegada més interès a poder escriure mòduls que facilitessin l'encapsulació i que, per tant, ajudessin a ocultar informació interna; així és com van sorgir nous llenguatges (per exemple, C ++) que donaven suport a aquesta tècnica i a nous paradigmes de programació més adequats. D'aquesta manera va aparèixer el paradigma de la programació orientada a objectes (POO).

2.4. Escalabilitat

L'herència i altres característiques del paradigma de la programació orientada a objectes permeten que l'esforç no augmenti exponencialment ni amb la grandària, ni amb la complexitat del projecte, ni amb un canvi d'especificacions.

2.5. Millor comprensió del codi

Tot i que ja hem dit que el paradigma de la programació estructurada millora la llegibilitat del codi respecte a no usar cap paradigma, no és suficient. Encara hi ha massa línies de codi i massa elements interrelacionats dispersos en diferents fitxers.

Gràcies a l'encapsulació, la POO millora la llegibilitat del codi. Degut al fet que les dades –les variables– i els procediments –funcions o mètodes– que conformen un objecte estan junts en un mateix element (anomenat *classe*), la comprensió del funcionament i la lògica del codi és més senzilla.

3. Programació orientada a objectes (POO)

3.1. Breu introducció als elements bàsics de la POO

A nivell molt introductori –ja ho veurem tot amb més detall–, cal destacar que l'element principal de la programació orientada a objectes (POO) és l'**objecte**. La POO modela la funcionalitat d'un programa –és a dir, del problema que cal resoldre– intentant assemblar-se el màxim possible a la realitat. La manera en què intenta imitar la realitat és definint un conjunt d'objectes que interactuen enviant i responent-se **missatges** els uns als altres. Gràcies a la col·laboració entre objectes, el programa és capaç de dur a terme la funcionalitat esperada –o de resoldre el problema plantejat.

D'igual manera que en el llenguatge C hi ha el tipus *integer* (`int`, sencer), *character* (`char`), etc. per a definir una variable, en els llenguatges que suporten la POO, el tipus dels objectes és una **classe**. Una classe s'assemblaria al que en programació estructurada anomenem *mòdul*, ja que una classe conté dades (en forma de variables) i funcions. Les dades d'una classe s'anomenen **atributs**, mentre que les seves funcions s'anomenen **mètodes**. D'igual manera, tant els atributs com els mètodes se solen anomenar **membres d'una classe**.

Així, doncs, quan definim un objecte, hem d'indicar-ne el tipus. Aquest no ha de ser ni un `int` ni un `char`, sinó una classe. De fet, moltes vegades, els objectes s'anomenen *instàncies d'una classe* o, simplement, **instàncies** (de l'anglès, *instance*), encara que la traducció correcta al català hauria de ser 'exemple' o 'cas'. Per tant, un objecte seria un exemple o cas particular d'una classe. És important entendre que podem tenir tants objectes o instàncies com vulguem d'una mateixa classe.

Com hem dit, les classes –que defineixen el tipus dels objectes– han de modelar la realitat del problema que estem tractant, la qual cosa ens obliga a definir les classes seguint una estratègia totalment diferent de la que seguim quan definim mòduls en la programació estructurada. Si en la programació estructurada els mòduls els definíem seguint un criteri, més o menys lògic, com agrupar funcions que tractin una mateixa tasca (p. ex., gràfics, control de regles, control de físiques, etc.), en la POO aquesta estratègia no ens servirà (del tot). Com ja hem dit anteriorment, la POO no utilitza un enfocament *top-down*, sinó *bottom-up*, és a dir, es parteix de la identificació dels objectes dins del problema per a abstreure les classes a les quals pertanyen.

Vegem a continuació un exemple senzill que ens permeti entendre els conceptes que han anat apareixent:

Exemple 4 (Conceptes bàsics de la POO) – La família POO

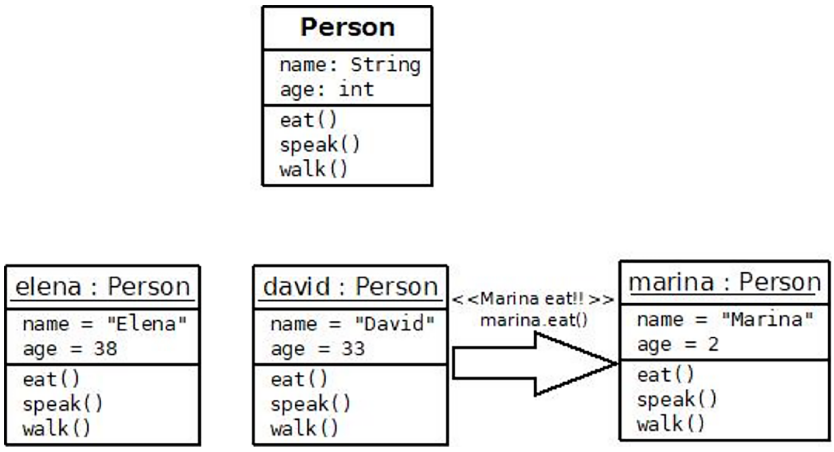
`Elena`, `Marina` i `David` són **objectes o instàncies** (exemples o casos) de la classe `Person`. Una persona (`Person`) té unes dades i uns comportaments, és a dir, uns **atributs** i uns **mètodes**. Alguns exemples d'atributs que totes les persones compartim són els següents: nom, cognom, data de naixement, lloc de naixement, gènere, raça, pes i alçada, entre altres. Mentre que exemples de mètodes que compartim totes les persones poden ser els següents: parlar, escoltar, caminar, menjar, augmentar l'alçada, augmentar el pes i reduir el pes, entre altres.

Una vegada que sabem que `Elena`, `Marina` i `David` són objectes de `Person`, és fàcil intuir que cadascun tindrà uns valors diferents per als atributs de la classe `Person`. Per exemple, el valor de l'atribut `nom` de la classe `Person` ja és diferent per a cadascun dels tres objectes, ja que un objecte té el valor `Elena`; un altre, `Marina`, i un altre, `David`. Així mateix, mentre que el valor de l'atribut gènere per a `David` és `home`, per a `Elena` i `Marina` el valor és `dona`.

D'igual manera, si `Marina` és un bebè de vint-i-cinc mesos, és fàcil imaginar que els valors per als atributs `alçada` i `pes` no poden ser iguals que els dels adults `Elena` i `David`; i que, al seu torn, els valors d'aquests atributs han de ser diferents entre `Elena` i `David`.

D'altra banda, com que `David` és el pare de `Marina`, a l'hora dels menjars aquest podria demanar a `Marina` que mengés enviant-li el **missatge** `"Marina, menja"` (és a dir, cridant el mètode `menjar` de `Marina`). Al seu torn, `Marina` podria respondre-li o bé menjant (per exemple, retornant el seu mètode `menjar` un `true` o un `1`) o bé negant-s'hi (per exemple, retornant un `false` o `0`). D'aquesta manera, veiem com interactuen (es comuniquen o col·laboren) els diferents objectes.

Podem veure aquest exemple de manera simplificada i visual en la figura següent:



El diagrama anterior està dibuixat seguint la notació UML –hi aprofundirem més endavant. Una classe està representada com una caixa amb tres parts: una conté el nom; l'altra, els atributs; i l'altra, els mètodes. El nom de la classe està escrit en negreta i alineada al centre de la caixa. Així mateix, hem indicat el tipus dels atributs, mentre que els mètodes els hem distingit obrint i tancant parèntesis al costat del seu nom.

Els objectes `Elena`, `David` i `Marina` també s'han representat amb una caixa dividida en tres parts, però amb el nom de la instància amb el format `nomInstancia:nomClasse`. Per als atributs hem indicat els valors que reben per a cadascun dels objectes. Així mateix, hem afegit una fletxa per mostrar un missatge de l'objecte `David` a l'objecte `Marina`. Aquest missatge crida el mètode `eat` de l'objecte `Marina`.

Reutilització

Una vegada codificada la classe `Person`, aquesta pot utilitzar-se en qualsevol programa que necessiti modelar una persona. Així, doncs, no cal tornar a reescriure codi, solament cal reutilitzar-lo.

Vídeo d'interès

Per acabar d'entendre els conceptes bàsics de la POO, us recomanem que vegeu el vídeo «Classe, objecte, atribut i mètode» que trobareu a l'aula de l'assignatura.

3.2. Donat un problema, quin és el procés que cal seguir?

La programació orientada a objectes (POO) intenta modelar el món real –o el problema real que volem resoldre– sobre la base de dos elements: els objectes i les classes. Per a això, el dissenyador del programa segueix un procés

d'abstracció *bottom-up* en el qual, a partir d'elements concrets (els objectes), intenta generalitzar fins a arribar a diferents plantilles o abstraccions (les classes). Concretament, el procés que se sol seguir és:

1) Identificar els objectes que interactuen en el problema. Donada una descripció del problema, els objectes solen ser substantius o sintagmes nominals.

2) Analitzar els objectes identificats amb l'objectiu de **reconèixer-ne les característiques i accions o funcionalitats**. Donada la descripció d'un problema, les característiques es corresponen amb substantius o sintagmes nominals, mentre que les accions se solen correspondre amb verbs o sintagmes verbals. En el paradigma de la programació orientada a objectes, les característiques s'anomenaran **atributs** i les accions, **mètodes**.

3) Identificar les classes que agrupen als diferents objectes. Cada classe ha de tenir les característiques (atributs) i accions o funcionalitats (mètodes) comunes dels objectes que representa.

4) Assignar cada objecte a les classes. Una vegada que tenim els objectes i les classes identificades, hem de relacionar els objectes o instàncies amb les classes corresponents. Un objecte pertany a una única classe.

Finalment, cal tenir present que no tots els substantius i verbs de la descripció del problema són objectes, atributs, mètodes o classes. Poden donar-se els casos següents:

- **Redundància (o sinònims).** Un objecte, classe, atribut o mètode és denominat de diferents maneres dins del text, és a dir, apareixen dos o més substantius o verbs per a indicar el mateix element. En aquests casos hem de quedar-nos amb una de les maneres de referir-nos a aquest element.

- **Irrellevància.** Pot donar-se el cas que algun element que hem identificat, per qualsevol motiu, tingui poc o res a veure amb el problema que volem solucionar, és a dir, que no ens sigui útil per al problema o context en el qual estem o, dit d'una altra manera, sigui irrellevant.

- **Rols.** Hi ha substantius i verbs que indiquen quin és el paper d'una classe o objecte dins del problema. De vegades, aquesta informació és supèrflua i, per tant, la podem descartar; altres vegades, els rols es corresponen amb atributs, operacions o relacions que és important tenir en compte.

Si ens fixem en l'Exemple 4 de l'apartat anterior, podem veure que hem seguit un procés *bottom-up*. Adonem-nos que hem anat de baix (objectes: `Elena`, `David` i `Marina`) a dalt (classe: `Person`). Concretament, primer hem identificat tres objectes –`Elena`, `David` i `Marina`–, i a partir d'aquests hem detectat que aquests tres objectes compartien una sèrie d'atributs i mètodes, la qual cosa

ens estava indicant que `Elena`, `David` i `Marina` podrien tractar-se del mateix tipus d'objecte, és a dir, que podrien pertànyer a la mateixa classe. Aquesta classe l'hem anomenada `Person`.

Algú podria pensar que es podria haver definit una classe per a `Elena`, una altra per a `David` i una altra per a `Marina`, però, en comptes d'això, hem fet un exercici d'abstracció i hem buscat elements comuns el resultat dels quals ha estat una classe comuna anomenada `Person`.

La identificació dels «elements compartits o comuns» requereix, per la nostra banda, exercir un nivell d'abstracció més gran del que solem exercir en el paradigma de la programació estructurada.

Vídeo d'interès

Per acabar d'entendre el concepte d'abstracció i l'enfocament *bottom-up*, us recomanem que vegeu el vídeo «Abstracció i paradigma bottom-up» que trobareu a l'aula de l'assignatura.

Aquesta manera de pensar un programa ens ajuda a fer un disseny més ben organitzat, més reduït quant a línies de codi, més reutilitzable i, per tant, més fàcil de mantenir i d'escalar.

A diferència de la programació estructurada, que se centra en les funcions, la POO se centra en l'estructura de les dades –és a dir, en com aquestes s'organitzen. Encara que pot semblar subtil, és vital que aquest canvi es comprengui i s'assimili per a tenir èxit amb la POO.

4. Com abordem la programació orientada a objectes?

És possible que estiguis pensant que això de la POO sembla complicat. Sentim dir-te que tens raó. Aprendre el paradigma POO i un llenguatge orientat a objectes concret és més complex que aprendre un llenguatge imperatiu lligat al paradigma de la programació estructurada. Així que, encara que no sigui senzill tot d'una, tampoc és impossible. Però tranquil, som conscients que ets un programador novell i, per això, anirem pas a pas. Com ja hem dit, la POO utilitza un enfocament *bottom-up*, diferent de l'enfocament *top-down* del paradigma de la programació estructurada. L'enfocament *bottom-up* se centra en abstraure els elements rellevants del domini de l'aplicació i, per desgràcia, la capacitat d'abstracció d'un programador novell acostuma a ser una de les seves febleses.

D'igual manera que anirem pas a pas, també et demanem que siguis conscient que dominar el paradigma de la POO t'exigirà un plus de compromís. En concret, t'exigirà i t'exigirem el següent:

- **Molta dedicació.** Passaràs desenes i desenes d'hores davant de l'ordinador programant, perquè, com se sol dir: «a programar, s'aprèn programant». Això és com aprendre a tocar la guitarra, si no dediques temps a la pràctica, ja et pots saber els acords de memòria, que no tocaràs cap cançó amb soltesa.

- **Tenir autonomia (també conegut com a «buscar-se la vida»).** Aquesta és una competència vital en ple segle XXI. Avui més que mai, la tecnologia evoluciona molt ràpidament i és molt difícil estar a l'última. L'equip docent no ho podem saber tot i, sovint, farem el mateix que pots fer tu: consultar Internet, un llibre, fer proves, etc. Hi ha milers de configuracions i, per si això fos poc, els programes solen tenir *bugs*. Així que, abans de preguntar alguna cosa, cerca pel teu compte la solució. La satisfacció de resoldre un problema per un mateix no té preu.

- **Compartir el teu coneixement.** Si resols un problema que t'ha mantingut en suspens dos dies, comparteix-ne la solució amb la resta de companys. T'ho agrairan. Recorda: «avui per tu, demà per mi». El mateix si saps la resposta a un dubte que es formula en un aula (sempre que no resolgui directament el que pregunta una activitat que cal lliurar). Explicar alguna cosa a algú és una oportunitat d'or per a aprofundir en el coneixement d'un tema, ja que moltes vegades un s'adona de si entén perfectament un concepte quan l'explica a una altra persona.

Resum

Punt de partida

- Donem per fet que coneixes i domines el paradigma de la programació estructurada abans de començar amb el paradigma de la programació orientada a objectes (POO).

Beneficis del paradigma de la programació orientada a objectes

- **Facilita la reutilització de codi** gràcies a l'encapsulació que força per si mateix una classe i l'herència.

- Un disseny del *software* **més abstracte** que aporta una solució més propera i natural a la realitat.

- **Protecció de les dades** d'un objecte mitjançant modificadors d'accés que garanteixen la consistència i faciliten el manteniment de les classes.

- Gràcies a la pròpia naturalesa del paradigma de la POO i als mecanismes que inclouen (p. ex. herència i encapsulació), és **més fàcil escalar un programa** sense que l'esforç de dur-ho a terme augmenti exponencialment.

- **Millora de la llegibilitat del codi** a causa que totes les dades i funcions o mètodes interrelacionats estan junts o encapsulats en una mateixa entitat *software* anomenada *classe*.

Elements bàsics del paradigma de la programació orientada a objectes

a) Classe:

- atributs
- mètodes

} membres d'una classe

b) Objecte o instància:

- un exemplar o cas concret d'una classe;

- una vegada creat, cada objecte ocuparà un espai de memòria i tindrà els seus propis atributs i mètodes que la classe a la qual pertany hagi definit.

c) Procés d'abstracció *bottom-up* per a dissenyar programes basats en la POO:

- **identificar** els diferents **objectes** del problema o context;
- **reconèixer** les **característiques** (dades; anomenades **atributs**) i **accions** (funcionalitats; anomenades **mètodes**) dels objectes identificats;
- agrupar els objectes per similitud per a **identificar** i **definir** les **classes** que els representen;
- **assignar** cada **objecte** a una de les **classes** definides.

Actitud davant del paradigma de la programació orientada a objectes

- **Molta dedicació:** durant hores hauràs de realitzar el procés iteratiu de pensar, programar, provar, buscar informació i entendre.
- **Autonomia:** buscar i provar diferents solucions als problemes que trobis.
- **Compartir coneixement:** aprenguem entre tots, compartim el que sabem o descobrim.

Activitats

Exercici 1
La lliga de baló presoner (LBP) està formada per deu equips com, per exemple, els Uocquis o els PAC Patrol. Cada equip té un nom, una data de fundació, un pressupost i una adreça on juguen els partits. Així, doncs, els Uocquis, creats el 6 d'octubre de 1995, tenen un pressupost de 5.000.000 € i juguen a l'avinguda Tibidabo de la ciutat de Barcelona. En canvi, els PAC Patrol tenen un pressupost de 16.000.000 € i juguen al carrer Tajo.

Evidentment, cada equip està format per jugadors. La normativa indica que cada jugador ha de tenir un dorsal i un nom que l'identifiqui. A més, tots els jugadors tenen un salari. Així, per exemple, el millor jugador de la lliga, Winnie Poo, juga amb el número 13 en els Rangers i cobra 100.000 € a l'any.

Segons la normativa, els equips solament poden fer ofertes als jugadors, fitxar-los quan aquests accepten les ofertes i vendre jugadors. Per la seva banda, els jugadors solament poden entrenar, jugar i acceptar o rebutjar les ofertes.

Segons les últimes notícies, els PAC Patrol han fet una oferta a Winnie Poo per fitxar-lo durant cinc anys i pagar-li 200.000 € a l'any. Els PAC Patrol estan en espera de rebre una resposta per part de Winnie Poo.

Atesa la descripció del problema anterior:

a) Indiqueu els objectes o instàncies que apareixen (solament el nom).

b) Indiqueu les classes que veieu (solament el nom).

c) Per cadascuna de les classes que heu esmentat en l'apartat «b», indiqueu-ne els atributs i mètodes segons el que diu el text.

d) Per a cada objecte identificat en l'apartat «a», indiqueu a quina classe de les esmentades en l'apartat «b» pertany.

e) Per a cada objecte indiqueu el valor que tenen els atributs de la classe a la qual pertany. Si per a un atribut no s'indica el valor en el text, poseu «Unknown».

f) Amb quin concepte de la POO es relaciona la frase en negreta?

Exercici 2
UocDonald's és una cadena de restaurants de menjar ràpid que té quinze sucursals repartides per tot el món. Cada sucursal té un identificador, la data en què va obrir les portes per primera vegada i l'adreça en la qual està situada. Així, doncs, el restaurant amb identificador 01, que va obrir per primera vegada el 8 de novembre de 2005, està situat al carrer Vila Olímpica, número 2, de la ciutat de Barcelona. En canvi, el restaurant 05 serveix menjars al carrer de Fuencarral, número 103, de Madrid.

L'especialitat de l'empresa són les hamburgueses. Per a cadascuna se'n sap el nom, el preu en euros, les quilocalories que aporta i si es pot demanar amb pa sense gluten o no. Per exemple, la Big Pac val 3,75 €, té 522 kcal i no és apta per a celíacs; no obstant això, la Bàsica val 1 €, té 254 kcal i sí en poden gaudir els celíacs.

La manera de treballar d'UocDonald's es basa en un model centralitzador, és a dir, l'empresa solament té un magatzem que proveeix totes les sucursals. Així, doncs, el magatzem central rep comandes de cadascuna de les sucursals i les envia a qui correspongui quan les té a punt. Per exemple, ahir mateix, el magatzem va rebre una comanda de la sucursal 01. És possible que en un futur es creïn més magatzems, però de moment solament n'hi ha un amb el nom de «Magatzem Central» i està situat a la població d'Ateca (Saragossa).

Donada la descripció del problema anterior:

a) Indiqueu els objectes o instàncies que apareixen (solament el nom).

b) Indiqueu les classes que veieu (solament el nom).

c) Per cadascuna de les classes que heu esmentat en l'apartat «b», indiqueu-ne els atributs i mètodes segons el que diu el text.

d) Per a cada objecte identificat en l'apartat «a», indiqueu a quina classe de les esmentades en l'apartat «b» pertany.

e) Per a cada objecte, indiqueu el valor que tenen els atributs de la classe a la qual pertany. Si per a un atribut no s'indica el valor en el text, poseu «Unknown».

Exercici 3
El museu MuseUOC vol guardar les seves obres d'art. Cada obra d'art té un nom, un autor, una data de creació, una data d'entrada al museu i una taxació (i.e. en quant està valorada l'obra).

Una de les obres més importants que té el MuseUOC és *El GuerniPAC*, un quadre realitzat per PACasso l'any 1994 i que va començar a formar part del museu a partir del 25 de setembre de 2013. Es calcula que el seu valor actual és de 200.000.000 €.

Una altra obra important és el retaule *El Jardí de les PS*, el valor del qual és de 75.000.000 € i és al museu des del 10 d'octubre de 2009.

El museu està compost per diverses sales. Cada sala té un nom, uns metres quadrats, una planta de l'edifici i un valor que indica si està oberta o tancada en aquest moment. Una d'elles és la «Sala Principal», de 300 m^2, situada a la planta 0; també hi ha la «Sala dels Sentits», de 100 m^2; i la «Sala de Restauració», de 500 m^2. Cada sala pot obrir-se o tancar-se, segons es vulgui. Evidentment, si és oberta no pot ser tancada i viceversa. Actualment, totes les sales són obertes.

Donada la descripció del problema anterior:

a) Indiqueu els objectes o instàncies que apareixen (solament el nom).

b) Indiqueu les classes que veieu (solament el nom).

c) Per cadascuna de les classes que heu esmentat en l'apartat «b», indiqueu-ne els atributs i mètodes segons el que diu el text.

d) Per a cada objecte identificat en l'apartat «a», indiqueu a quina classe de les esmentades en l'apartat «b» pertany.

e) Per a cada objecte indiqueu el valor que tenen els atributs de la classe a la qual pertany. Si per a un atribut no s'indica el valor en el text, poseu «Unknown».

Exercici 4
La bibliUOCteca (biblioteca de la UOC) vol desar tot l'inventari dels llibres de què disposa. Cada llibre té un títol, un autor, una data de creació, una data d'adquisició i un preu (i.e. el que va costar adquirir-lo).

Un dels llibres més importants que té la bibliUOCteca és *L'ombra de la PS*, escrit per Carlos Ruiz PACfón l'any 2001 i adquirit, el 8 de novembre de 2013, per 21,99 €.

Un altre llibre important és *PSity*, que es va comprar per 18 € i és a la biblioteca des del 3 d'octubre de 2008.

La biblioteca està composta per diverses sales. Cada sala té un nom, uns metres quadres, una planta de l'edifici i un valor que indica si és oberta o tancada en aquest moment. Una d'elles és la «Sala d'Actes», de 300 m^2, situada a la planta 0. També hi ha la «Sala d'Estudi», de 150 m^2. Cada sala pot obrir-se o tancar-se segons es vulgui. Evidentment, si és oberta no pot ser tancada i viceversa. Actualment, totes les sales són obertes.

Donada la descripció del problema anterior:

a) Indiqueu els objectes o instàncies que apareixen (solament el nom).

b) Indiqueu les classes que veieu (solament el nom).

c) Per cadascuna de les classes que heu esmentat en l'apartat «b», indiqueu-ne els atributs i mètodes segons el que diu el text.

d) Per a cada objecte identificat en l'apartat «a», indiqueu a quina classe de les esmentades en l'apartat «b» pertany.

e) Per a cada objecte indiqueu el valor que tenen els atributs de la classe a la qual pertany. Si per a un atribut no s'indica el valor en el text, poseu «Unknown».

Solucionari

Solució exercici 1

a) Indiqueu els objectes o instàncies que apareixen (solament el nom).

- Uocquis
- PAC Patrol
- Winnie Poo
- Rangers

L'LBP no la considerem un objecte, ja que el problema que estem modelant o solucionant és la mateixa LBP, per tant, no té sentit.

b) Indiqueu les classes que veieu (solament el nom).

- Equip
- Jugador

c) Per cadascuna de les classes que heu esmentat en l'apartat «b», indiqueu-ne els atributs i mètodes segons el que diu el text.

Classe	Equip	Jugador
Atributs	Nom Data de fundació Pressupost Adreça	Dorsal Nom Salari
Mètodes	Fer ofertes jugadors (ferOferta) Fitxar jugadors (fitxarJugador) Vendre jugadors (vendreJugador)	Entrenar (entrenar) Jugar (jugar) Acceptar o rebutjar oferta (acceptarRebutjarOferta)

A `Equip` es podria considerar el mètode `jugar`. Tant si s'ha posat com si no, és correcte. Si el mètode `acceptarRebutjarOferta` s'ha dividit en dos (p. ex., `acceptarOferta` i `rebutjarOferta`), es pot considerar correcte.

Per a ser del tot correctes i modelar la frase o relació esmentada en el text «cada equip està format per jugadors», hauríem d'haver posat un atribut tipus `Jugador` en la classe `Equip`. Aquest atribut de la classe `Equip` seria un *array* (o similar). Si ho heu posat, perfecte, ja que vol dir que esteu anant un pas més enllà del que hem explicat fins ara. Si no ho heu posat, tranquils, ja que la relació entre objectes (denominada formalment *associació*) s'explicarà en el mòdul «Associacions (relacions entre objectes)» d'aquests materials docents. Ara com ara, solament treballem les classes de manera aïllada, és a dir, sense relacionar-les entre elles.

d) Per a cada objecte identificat a l'apartat «a», indiqueu a quina classe de les esmentades en l'apartat «b» pertany.

`Uocquis` és un objecte que pertany a la classe `Equip`.

`PAC Patrol` és un objecte que pertany a la classe `Equip`.

`Winnie Poo` és un objecte que pertany a la classe `Jugador`.

`Rangers` és un objecte que pertany a la classe `Equip`.

e) Per a cada objecte, indiqueu el valor que tenen els atributs de la classe a la qual pertany. Si per a un atribut no s'indica el valor en el text, poseu «Unknown».

Atribut/Objecte	Uocquis	PAC Patrol	Rangers
Nom	Uocquis	PAC Patrol	Rangers
Data de fundació	6 d'octubre de 1995	«Unknown»	«Unknown»
Pressupost	5.000.000 €	16.000.000 €	«Unknown»

Atribut/Objecte	Uocquis	PAC Patrol	Rangers
Adreça	Avinguda Tibidabo (Barcelona)	Carrer Tajo	«Unknown»

Atribut/Objecte	Winnie Poo
Dorsal	13
Nom	Winnie Poo
Salari	100.000 €

f) Amb quin concepte de la POO es relaciona la frase en negreta?

Fa referència al concepte de missatge. La versió codificada del text podria ser:

```
pecpatrol.ferOferta(WinniePoo, 5, 200.000);
```

Solució exercici 2

a) Indiqueu els objectes o instàncies que apareixen (solament el nom).

- Sucursal 01
- Sucursal 05
- Big Pac
- Bàsica
- Magatzem Central

b) Indiqueu les classes que veieu (solament el nom).

- Sucursal (o també el podíeu haver anomenat «Restaurant»)
- Hamburguesa
- Magatzem

c) Per cadascuna de les classes que heu esmentat en l'apartat «b», indiqueu-ne els atributs i mètodes segons el que diu el text.

Classe	Sucursal	Hamburguesa	Magatzem
Atributs	Identificador Data d'obertura Adreça	Nom Preu Kcal senseGluten	Nom Adreça (o població)
Mètodes			Rebre comanda de sucursal (rebreComanda) Enviar comanda a sucursal (enviarComanda)

El mètode `rebreComanda` podria haver-se considerat com `ferComanda` i haver-lo posat en la classe `Sucursal`. Igualment, es podria haver creat el mètode `rebreComanda` (de magatzem) a la classe `Sucursal`. En qualsevol cas, la relació entre les sucursals i el magatzem hauria de quedar reflectida en la solució d'una manera o una altra.

Les adreces de les sucursals es podrien haver separat en tres atributs: carrer, número i població.

d) Per a cada objecte identificat en l'apartat «a», indiqueu a quina classe de les esmentades en l'apartat «b» pertany.

`Sucursal01` és un objecte que pertany a la classe `Sucursal`.

`Sucursal05` és un objecte que pertany a la classe `Sucursal`.

`Big Pac` és un objecte que pertany a la classe `Hamburguesa`.

`Bàsica` és un objecte que pertany a la classe `Hamburguesa`.

`Magatzem Central` és un objecte que pertany a la classe `Magatzem`.

e) Per a cada objecte indiqueu el valor que tenen els atributs de la classe a la qual pertany. Si per a un atribut no s'indica el valor en el text, poseu «Unknown».

Atribut/ Objecte	Sucursal01	Sucursal05
Identificador	01	05
Data d'obertura	8 de novembre de 2005	«Unknown»
Adreça	Carrer Vila Olímpica, número 2 (Barcelona)	Carrer de Fuencarral, número 103 (Madrid)

Atribut/Objecte	Big Pac	Bàsica
Nom	Big Pac	Bàsica
Preu	3,75 €	1 €
Kcal	522	254
senseGluten	No (false)	Sí (true)

Atribut/Objecte	Magatzem Central
Nom	Magatzem Central
Adreça	Ateca (Saragossa)

Solució exercici 3

a) Indiqueu els objectes o instàncies que apareixen (solament el nom).

- El GuerniPAC
- El Jardí de les PS
- Sala Principal
- Sala dels Sentits
- Sala de Restauració

b) Indiqueu les classes que veieu (solament el nom).

- Obra
- Sala

c) Per cadascuna de les classes que heu esmentat en l'apartat «b», indiqueu-ne els atributs i mètodes segons el que diu el text.

Classe	Obra	Sala
Atributs	Nom Autor Data de creació Data d'entrada Taxació	Nom m^2 Planta esOberta
Mètodes		obrirTancar()

El mètode `obrirTancar()` podria haver-se fet amb dos mètodes: `obrir()` i `tancar()`. No obstant això, en tractar-se de dos mètodes que assignen el valor contrari a l'atribut `esOberta`, es pot fer tot amb un únic mètode que assigna a l'atribut `esOberta` el valor contrari al que tingui en aquest moment.

d) Per a cada objecte identificat en l'apartat «a», indiqueu a quina classe de les esmentades en l'apartat «b» pertany.

`El GuerniPAC` és un objecte que pertany a la classe `Obra`.

`El Jardí de les PS` és un objecte que pertany a la classe `Obra`.

`Sala Principal` és un objecte que pertany a la classe `Sala`.

`Sala dels Sentits` és un objecte que pertany a la classe `Sala`.

`Sala de Restauració` és un objecte que pertany a la classe `Sala`.

e) Per a cada objecte, indiqueu el valor que tenen els atributs de la classe a la qual pertany. Si per a un atribut no s'indica el valor en el text, poseu «Unknown».

Atribut/Objecte	El GuerniPAC	El Jardí de les PS
Nom	El GuerniPAC	El Jardí de les PS
Autor	PACasso	«Unknown»
Data de creació	1994	«Unknown»
Data d'entrada	25 de setembre de 2013	10 d'octubre de 2009
Taxació	200.000.000 €	75.000.000 €

Atribut/Objecte	Sala Principal	Sala dels Sentits	Sala de Restauració
Nom	Sala Principal	Sala dels Sentits	Sala de Restauració
m^2	300	100	500
Planta	0	«Unknown»	«Unknown»
esOberta	Sí (true)	Sí (true)	Sí (true)

Solució exercici 4

a) Indiqueu els objectes o instàncies que apareixen (solament el nom).

- L'ombra de la PS
- PSity
- Sala d'Actes
- Sala d'Estudi

b) Indiqueu les classes que veieu (solament el nom).

- Llibre
- Sala

c) Per cadascuna de les classes que heu esmentat en l'apartat «b», indiqueu-ne els atributs i mètodes segons el que diu el text.

Classe	Llibre	Sala
Atributs	Títol Autor Data de creació Data d'adquisició Preu	Nom m^2 Planta esOberta
Mètodes		obrirTancar()

El mètode `obrirTancar()` podria haver-se fet amb dos mètodes: `obrir()` i `tancar()`. No obstant això, en tractar-se de dos mètodes que assignen el valor contrari a l'atribut `esOberta`, es pot fer tot amb un únic mètode que assigna a l'atribut `esOberta` el valor contrari al que tingui en aquest moment.

d) Per a cada objecte identificat en l'apartat «a», indiqueu a quina classe de les esmentades en l'apartat «b» pertany.

`L'ombra de la PS` és un objecte que pertany a la classe `Llibre`.

`PSity` és un objecte que pertany a la classe `Llibre`.

`Sala d'Actes` és un objecte que pertany a la classe `Sala`.

`Sala d'Estudi` és un objecte que pertany a la classe `Sala`.

e) Per a cada objecte, indiqueu el valor que tenen els atributs de la classe a la qual pertany. Si per a un atribut no s'indica el valor en el text, poseu «Unknown».

Atribut/Objecte	L'ombra de la PS	PSity
Títol	L'ombra de la PS	PSity
Autor	Carlos Ruiz PACfón	«Unknown»
Data de creació	2011	«Unknown»
Data d'adquisició	8 de novembre de 2013	3 d'octubre de 2008
Preu	21,99 €	18 €

Atribut/Objecte	Sala d'Actes	Sala d'Estudi
Nom	Sala d'Actes	Sala d'Estudi
m^2	300	150
Planta	0	«Unknown»
esOberta	Sí (true)	Sí (true)

Bibliografia

Booch, G.; Maksimchuk, R.; Engle, M.; Young, B. J.; Conallen, J.; Houston, K. (2007). *Object-Oriented Analysis and Design with Applications*. Addison-Wesley Professional. ISBN: 978-0201895513.

Griffiths, D.; Griffiths, D. (2019). *Head First Kotlin*. O'Reilly Media, Inc. ISBN: 978-1491996690.

Hunt, A.; Thomas, D. (2019). *The Pragmatic Programmer: your journey to mastery, 20th Anniversary Edition* (2a. ed.). Addison-Wesley Professional. ISBN: 978-0135956977.

Phillips, D. (2018). *Python 3 Object-Oriented Programming* (3a. ed.). Packt Publishing. ISBN: 978-1789615852.

Pollice, G.; West, D.; McLaughlin, B. (2006). *Head First Object-Oriented Analysis and Design*. O'Reilly Media, Inc. ISBN: 978-0596008673.

Robinson, S.; Nagel, C.; Watson, K.; Glynn, J.; Skinner, M.; Evjen, B. (2004). *Professional C#* (3a. ed.). Wrox. ISBN: 978-0764557590.

Sharp, J. (2007). *Microsoft Visual C# 2008 Step by Step*. Microsoft Press. ISBN: 978-0735624306.

Weisfeld, M. (2019). *The Object-Oriented Thought Process* (5a. ed.). Addison-Wesley Professional. ISBN: 978-0135182130.