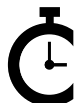
Associacions (relacions entre objectes)

PID_00269653

David García Solórzano

Temps mínim de dedicació recomanat: 3 hores



Universitat Oberta de Catalunya

David García Solórzano

Graduat Superior en Enginyeria de 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 2020
© David García Solórzano
Tots els drets reservats
© d'aquesta edició, FUOC, 2020
Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Cap part d'aquesta publicació, incloent-hi el disseny general i la coberta, no pot ser copiada, reproduïda, emmagatzemada o transmesa de cap manera ni per cap mitjà, tant si és elèctric com químic, mecànic, òptic, de gravació, de fotocòpia o per altres mètodes, sense l'autorització prèvia per escrit dels titulars dels drets.

Índex

Introducció 5

Objectius 6

1. Associacions 7
- 1.1. Associació binària 7
- 1.2. Associació d'agregació 8
- 1.3. Associació de composició 9
- 1.4. Associació reflexiva 10
- 1.5. Propietats de les associacions 10
  - 1.5.1. Multiplicitat 11
  - 1.5.2. Navegabilitat 13
  - 1.5.3. Rol 14
- 1.6. Traduint a codi 15
  - 1.6.1. Associacions binàries, reflexives i d'agregació 16
  - 1.6.2. Associació de composició 20
- 1.7. Dos objectes poden estar relacionats entre ells més d'una vegada? 21

2. Classe associativa 22
- 2.1. Explicació 22
- 2.2. Traduint a codi 23

3. Classe parametritzada 24
- 3.1. Explicació 24
- 3.2. Traduint a codi 26

Resum 27

Bibliografia 29

Introducció

Fins ara hem vist una classe –formada pels seus atributs i mètodes– treballar de manera aïllada. És obvi que una única classe no pot solucionar per si sola un problema i que, per tant, ha de col·laborar amb altres classes per a resoldre'l. En aquest mòdul veurem com els objectes de diferents classes poden col·laborar o relacionar-se entre ells. Principalment veurem:

- associacions
- classe associativa
- classe parametritzada

A més, veurem com es codifiquen i es representen aquestes relacions usant el llenguatge de modelatge de sistemes de programari UML (*Unified Modeling Language*).

Tret que s'indiqui un llenguatge de programació concret, els exemples de codificació estan escrits amb un llenguatge de programació inventat, és a dir, un pseudocodi. Si haguéssim de dir a quin llenguatge de programació real s'assembla el pseudocodi que emprem, diríem que és semblant a Java (però sense elements que dificultin la comprensió dels exemples).

Així doncs, haurem de consultar la documentació del llenguatge de programació que volem utilitzar per a veure com es codifiquen els conceptes explicats i els exemples proporcionats.

Objectius

L'objectiu principal d'aquest mòdul és explicar com els objectes es relacionen entre ells dins d'un programa basat en el paradigma de la programació orientada a objectes. Com a conseqüència d'aquest objectiu, n'apareixen d'altres:

1. Entendre què és una associació i distingir els quatre tipus més importants: binària, agregació, composició i reflexiva.

2. Conèixer les tres propietats bàsiques que tenen les associacions: multiplicitat, navegabilitat i rol.

3. Entendre què és una classe associativa i saber quan el seu ús és necessari.

4. Comprendre què és una classe parametritzada i en quins contextos s'acostuma a utilitzar.

5. Saber representar en un diagrama de classes UML els quatre tipus d'associació vistos en aquest mòdul, la classe associativa i la classe parametritzada.

6. Saber i entendre com es codifiquen els quatre tipus d'associació que s'expliquen en aquest mòdul, com també la classe associativa i la classe parametritzada.

1. Associacions

La relació entre objectes o instàncies és coneguda amb el nom d'*associació*. Podem distingir principalment quatre tipus d'associacions:

- associació binària
- associació d'agregació
- associació de composició
- associació reflexiva

Veurem aquests quatre tipus a continuació.

1.1. Associació binària

Podem definir l'**associació binària** (*binary association*) com la relació entre instàncies/objectes de dues classes en què els objectes d'una classe existeixen de manera independent de l'existència dels objectes de l'altra classe.

Nota

És important destacar que els objectes de dues classes poden relacionar-se de manera diferent segons el problema o context en el qual es troben i el criteri del dissenyador del programa.

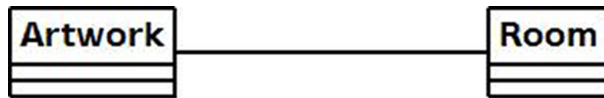
Així doncs, la creació o destrucció d'una instància de la classe `A` implica únicament la creació o destrucció de la relació que hi ha entre aquesta instància i una altra instància de la classe `B`, però mai no significa la creació o destrucció de la instància de la classe `B`.

Parlant amb propietat, es diu que no hi ha una relació forta entre totes dues instàncies. L'associació binària respon a una relació que indica que l'objecte de la classe `A` *usa* un objecte de la classe `B` i pot ser que viceversa també, com ja veurem.

Exemple 1 (associació binària) – Les obres d'art i les sales d'un museu

Imagina un museu que conté obres d'art. Cada obra d'art (instància o objecte de la classe `Artwork`) està exposada en una sala del museu (instància o objecte de la classe `Room`). Ara pensem en la relació que hi ha entre els objectes de totes dues classes. Si es destrueix una obra d'art (per exemple, es crema o es deteriora), això no significa que destruïm la sala del museu (quina gràcia!), ja que pot haver-hi més obres d'art exposades. El mateix passa a l'inrevés: si destruïm una sala perquè la volem fusionar amb una altra o volem usar-ne l'espai per a una altra cosa –per exemple, ja no serà una sala (`Room`), sinó un despatx (`Office`)–, les obres d'art que alberga no les hem de destruir (només faltaria!), en tot cas, les haurem de ressituar (reassignar) a altres sales del museu.

En UML, aquesta associació es representa amb una línia que uneix totes dues classes:



1.2. Associació d'agregació

Podem definir l'**associació d'agregació** (*aggregation* o *shared aggregation*) com la relació entre instàncies de dues classes quan una d'aquestes (anomenada *component*) és part de l'altra (anomenada *compost*).

Nota

Per a facilitar la comprensió i reduir l'extensió d'aquests materials, s'han eliminat els atributs i mètodes de les classes dels diagrames de classes.

Aquest tipus d'associació també és conegut com a **composició feble**. S'assumeix una subordinació conceptual del tipus «tot/part», «està format per», o bé «té un». Així doncs, es tracta d'un cas particular d'associació binària en la qual hi ha una certa relació d'assemblatge, física o lògica, entre les dues instàncies.

Una de les característiques de l'agregació és que la **destrucció del compost no significa la destrucció dels components, ni viceversa**. A més, l'**objecte component pot estar present en més d'una associació d'agregació**.

Exemple 2 (associació d'agregació) – Les obres d'art i les col·leccions

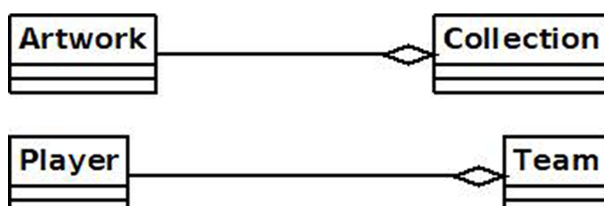
En molts museus, les obres d'art (instància o objecte de la classe `Artwork`) pertanyen a una col·lecció (instància o objecte de la classe `Collection`). Ara pensem en la relació que hi ha entre els objectes de totes dues classes. Podem veure que una col·lecció està formada per obres d'art. Així doncs, la col·lecció és el tot (classe composta) i les obres d'art són la part (classe component). A més, si es destrueix una obra d'art (per exemple, es crema o es deteriora) o simplement decidim que ja no pertany a la col·lecció que té assignada, això no significa que destruïm (que ja no existeixi) la col·lecció a la qual pertanyia fins ara, ja que pot haver-hi més obres d'art que sí hi pertanyin. Passa el mateix a l'inrevés: si decidim que ja no volem englobar les obres d'art sota el paraigua d'una col·lecció, eliminem la col·lecció, però no les obres d'art que en formaven part. Finalment, cal dir que una obra d'art pot pertànyer a diverses col·leccions alhora.

Vegem un altre exemple:

Exemple 3 (associació d'agregació) – L'equip de bàsquet

Un equip de bàsquet (instància o objecte de la classe `Team`) està format per jugadors o té jugadors (instància o objecte de la classe `Player`). L'equip és el compost, i cada jugador és un component. Si eliminem un jugador de l'equip (per exemple, l'hem transferit o s'ha retirat), l'equip no desapareix, òbviament. Així mateix, si l'equip desapareix (per exemple, entra en números vermells), els jugadors no desapareixen, se'n van a altres equips (formaran part d'altres equips).

La manera de representar en UML una associació d'agregació és mitjançant un rombe blanc en l'extrem de la classe dels objectes que fan de compost.



1.3. Associació de composició

Podem definir l'**associació de composició** (*composition* o *composite aggregation*) com un cas particular de l'associació d'agregació en la qual la vida o existència de la instància de la classe component depèn de l'existència de la instància de la classe composta. Així doncs, l'objecte component solament pot existir si existeix l'objecte compost. Això comporta que la destrucció de l'objecte compost suposa la destrucció de tots els seus components, però no a l'inrevés.

Una altra de de les característiques que diferencien la composició de l'agregació és que **en la composició, cada component solament pot relacionar-se mitjançant composició amb un únic compost**. Per tot això, aquest tipus d'associació també es coneix com a **composició forta**. No obstant això, un objecte component pot relacionar-se amb objectes d'altres classes mitjançant altres tipus d'associació.

Exemple 4 (associació de composició) – La mà

Mira una de les teves mans. Està formada per dits, oi? En aquest punt podem dir que la relació entre mà (objecte de la classe `Hand`) i els dits (objectes de la classe `Finger`) és de compost i components. És una associació d'agregació o de composició? Si amputem la mà, implícitament eliminem els dits que la componen, però si amputem un dit de la mà, no amputem tota la mà! A més, els dits solament existeixen per a aquesta mà. Això ens fa veure que perquè existeixin els dits, ha d'existir la mà que els conté; per tant, som al davant d'un cas de composició.

Ocorre el mateix amb l'exemple següent:

Exemple 5 (associació de composició) – La cadira

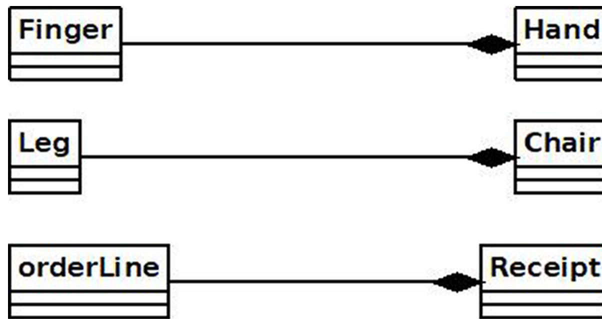
Una cadira (objecte de la classe `Chair`) està composta o formada per potes (objectes de la classe `Leg`). Si destruïm la cadira, n'estem destruint les potes implícitament. Si llevem una pota a la cadira, la cadira continua existint. Estem clarament davant un cas de composició.

Finalment:

Exemple 6 (associació de composició) – El tiquet de compra

El tiquet de compra que ens donen en un supermercat (objecte de la classe `Receipt`) està format per línies de compra (objectes de la classe `orderLine`) que ens indiquen el producte que hem adquirit, la quantitat i el preu. Cada línia del tiquet solament existeix en aquest tiquet i perquè el tiquet existeix. Si perdem el tiquet, perdem les línies de compra. Si traiem una línia de compra (per exemple, la ratllem), el tiquet no desapareix. Així doncs, es tracta d'una associació de composició.

La composició es representa igual que l'agregació, però el rombe ara és de color negre.



1.4. Associació reflexiva

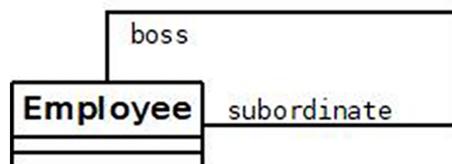
Podem definir l'associació **reflexiva** (*reflexive association*) com una associació binària en la qual els objectes que participen en la relació pertanyen a la mateixa classe, és a dir, els objectes origen i destinació són de la mateixa classe.

Vegem-ne un exemple:

Exemple 7 (associació reflexiva) – El cap i els seus subordinats

Pensa en qualsevol feina. Sempre hi ha un empleat que és cap d'uns altres. Així doncs, tant el cap com els treballadors al seu càrrec són empleats. Això sí, cada empleat existeix independentment de la resta, i un empleat no està format o compost per altres empleats.

La manera de representar una associació reflexiva és igual que la manera de representar l'associació binària (no deixa de ser un cas particular d'aquesta), tret que la línia surt i entra a la mateixa classe.



Com pots veure en el diagrama de classes anterior, hi ha dues etiquetes o textos: `<<boss>>` i `<<subordinate>>`. Aquestes etiquetes es diuen **rols** (ho veurem en aquest mòdul) i s'escriuen en els extrems de l'associació per a identificar correctament i semànticament quin paper té cada objecte dins de la relació.

1.5. Propietats de les associacions

Les associacions anteriors tenen bàsicament tres propietats:

- multiplicitat
- navegabilitat
- rol

A continuació les veurem una per una.

1.5.1. Multiplicitat

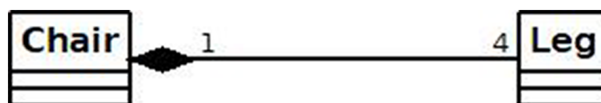
A causa que una associació entre dues classes representa en realitat una relació entre instàncies o objectes d'aquestes dues classes, hem d'indicar quantes instàncies de cada classe estan involucrades en la relació. Aquest nombre d'instàncies que participen en la relació és, precisament, la multiplicitat de l'associació.

Definició de multiplicitat

És el nombre mínim i màxim d'instàncies d'una classe amb el qual es pot relacionar una instància de l'altra classe de la relació.

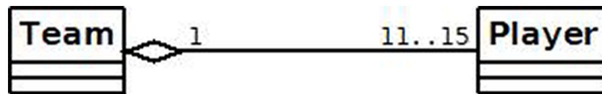
La manera d'indicar la multiplicitat és mitjançant modificadors que s'afegeixen al damunt o a sota de la línia que representa la relació i prop de la classe. Cada associació té dues multiplicitats, una per extrem, és a dir, una per cada classe que participa en l'associació. Com hem dit, cada multiplicitat indica un nombre mínim (anomenat *opcionalitat* o *participació*) i màxim (anomenat *cardinalitat*) d'objectes d'aquesta classe que es relacionen amb un objecte de l'altra classe. Així doncs, la multiplicitat d'una classe està formada per la participació (valor mínim) i la cardinalitat (valor màxim). Hi ha diferents maneres d'indicar la multiplicitat.

1) Nombre exacte: indica que una instància de la classe `A` ha d'estar relacionada exactament amb el nombre indicat d'instàncies de la classe `B`. Així doncs, la participació i la cardinalitat que conformen la multiplicitat de la classe coincideixen.

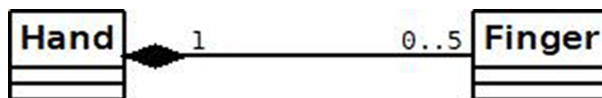


En l'exemple anterior, un objecte de tipus `Chair` ('cadira') es relaciona amb quatre objectes de la classe `Leg` ('pota'). En aquest cas, la cardinalitat i la participació coincideixen essent `4`. Així mateix, un objecte de tipus `Leg` es relaciona amb un únic objecte de tipus `Chair` (lògic en tractar-se d'una composició). En aquest cas, també la cardinalitat i la participació coincideixen, però en el valor `1`.

2) Rang de valors: permet indicar el nombre mínim i màxim d'instàncies de la classe `B` amb les quals un objecte de la classe `A` ha de relacionar-se.

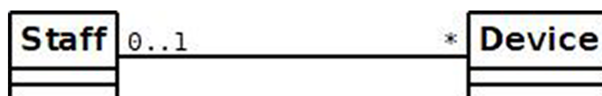


Segons la normativa de l'NBA, un equip que juga en aquesta lliga de bàsquet ha de tenir entre 11 i 15 jugadors. Així doncs, no en pot tenir ni més ni menys. En aquest cas, la multiplicitat de la classe Player és 11..15, essent la participació 11 i la cardinalitat 15. Per la seva banda, un jugador solament pot pertànyer a un equip. Com que la participació és 1, no existeix l'opció que un jugador no tingui equip (quan s'usa el rang 0..1, s'està dient que l'objecte pot relacionar-se o no, per aquest motiu la participació –el valor mínim de la multiplicitat– també es coneix com a *opcionalitat*).



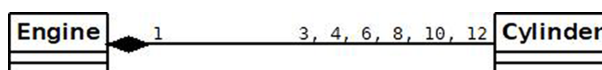
Una mà humana pot tenir entre 0 i 5 dits. Un dit pertany a una única mà. En aquest cas és opcional que una mà tingui dits, pot ser que en no tingui cap; per aquest motiu indiquem que la participació és 0 dins de la multiplicitat de la classe Finger.

3) Símbol * : existeix la possibilitat d'utilitzar el símbol * (asterisc) per a indicar que el nombre d'instàncies és indefinit, és a dir, amb * estem dient que un objecte de la classe A pot estar relacionat amb qualsevol nombre d'instàncies de la classe B, incloent cap (zero). Escriure * és equivalent a indicar la multiplicitat 0..*.



L'exemple anterior indica que un membre de l'equip (Staff) pot tenir o usar entre 0 (cap) i un nombre indeterminat de dispositius (Device); p.ex. un ordinador de taula, un portàtil, un mòbil, etc. A més, cada dispositiu només el pot tenir o utilitzar una persona alhora com a màxim (també pot no utilitzar-lo ningú).

4) Rangs no consecutius: separant nombres exactes o rangs per comes, es poden fer rangs no consecutius.



Un motor (`Engine`) pot tenir 3, 4, 6, 8, 10 o 12 cilindres (però no 9, per exemple), i un cilindre (`Cylinder`) solament pot pertànyer a un motor. Aquesta relació és una composició, ja que la destrucció de l'objecte `Engine` ha de suposar la destrucció dels objectes `Cynlinder`, ja que l'existència dels objectes `Cylinder` depèn de l'existència d'un objecte `Engine` que els contingui.

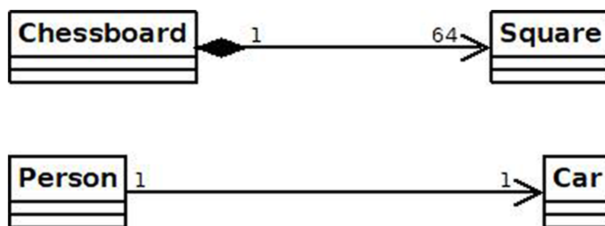
Multiplicitat en la composició

Atès que l'objecte component en una associació de composició solament pot existir si existeix l'objecte compost al qual pertany, i no pot ser part d'una altra composició, la multiplicitat a la classe que fa de compost (en què està el rombe negre) és `0..1` o `1`. El cas `0` és una mica especial, i vol dir que si l'objecte component és extret de la composició abans d'eliminar l'objecte compost, llavors l'objecte component que ha estat extret pot continuar existint. No obstant això, si no s'ha extret l'objecte component de l'objecte compost i s'elimina aquest últim, llavors l'objecte component també s'elimina. Per tant, el més freqüent és una multiplicitat igual a `1` al costat del compost.

1.5.2. Navegabilitat

Una altra propietat que podem expressar en una associació dins d'un diagrama de classes és la seva navegabilitat o direccionalitat, és a dir, podem indicar si una instància d'una classe coneix l'existència d'una instància de l'altra classe. Bàsicament hi ha dos tipus de navegabilitat:

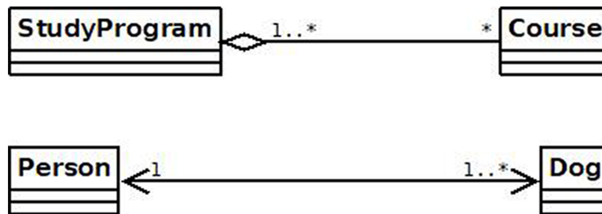
1) Unidireccional: solament una de les dues instàncies té constància de l'existència de l'altra. La navegabilitat unidireccional es representa amb una punta de fletxa que apunta a la classe que no té constància.



En el primer dels dos exemples podem veure com un tauler d'escacs (`Chessboard`) està compost per seixanta-quatre quadrats (`Square`), i un quadrat solament pertany a un tauler. És el tauler (`Chessboard`) el que sap de l'existència dels seixanta-quatre quadrats (`Square`), és a dir, donat un tauler, podem conèixer-ne els seixanta-quatre quadrats, però no a l'inrevés.

En el segon exemple, una persona (`Person`) condueix un cotxe (`Car`), i un cotxe solament pot ser conduït per una persona en un moment determinat. En aquest cas, la navegabilitat ens està dient que, donat un objecte de tipus `Person`, podem saber quin cotxe està conduint; no obstant això, donat un objecte `Car`, no podem saber quina persona el condueix.

2) Bidireccional: les dues instàncies tenen constància de l'existència de l'altra. La navegabilitat bidireccional es representa sense puntes de fletxa. També pots veure en alguns diagrames de classes que la bidireccionalitat està representada dibuixant puntes de fletxa en tots dos costats de la línia (mira l'exemple `Person-Dog`). Això és correcte, encara que és menys habitual.



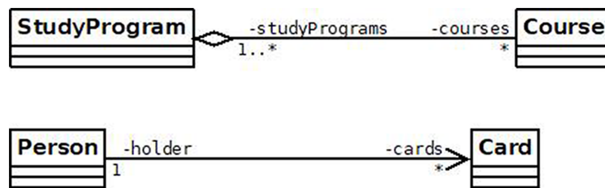
En el primer exemple, `StudyProgram-Course`, un pla d'estudis (`StudyProgram`) està format per diferents assignatures (`Course`). En aquest cas, es preveu el cas que un pla d'estudis no estigui format per cap assignatura. Una assignatura, al seu torn, pot formar part d'un o diversos plans d'estudis. Per exemple, l'assignatura de *Disseny i programació orientada a l'objecte* forma part dels plans d'estudi dels graus d'Enginyeria Informàtica, Tecnologies de Telecomunicació, Multimèdia i Ciència de Dades Aplicades, entre d'altres. Gràcies a la navegació bidireccional, donat un pla d'estudis, podem saber-ne les assignatures i, donada una assignatura, podem saber a quins plans d'estudi pertany.

En el segon exemple estem dient que una persona (`Person`) té un o més gossos (`Dog`), i que un gos té solament una persona com a propietari. Donada una persona, sabem quins gossos té, i donat un gos, podem saber qui n'és l'amo.

1.5.3. Rol

Aquesta propietat és una mica diferent de les dues anteriors. A diferència de la multiplicitat i la navegabilitat, el rol no modifica el comportament de les associacions, sinó que afegeix informació contextual que ha d'ajudar el lector a comprendre la relació.

El rol no és més que una etiqueta que es posa a cada costat de l'associació per a anomenar les classes participants d'una manera diferent o, dit d'una altra manera, per a indicar el paper que té cada classe. El rol té rellevància quan traduïm una associació del diagrama de classes a codi. Concretament, cada rol es convertirà en un atribut en la classe coneixedora i serà del tipus de l'altra classe. És per això molt habitual anteposar al nom del rol el símbol corresponent al modificador d'accés que volem, és a dir: `+`, `-`, `#` o `~`. Si no es posa, s'entén que és privat.



En el primer exemple, la classe `StudyProgram` tindrà un atribut anomenat `courses` que serà privat (fixem-nos en el símbol – precedint el nom del rol). Aquest atribut serà un *array* (o una col·lecció) que emmagatzemarà objectes de la classe `Course` (pot estar buit). Així mateix, la classe `Course` tindrà un atribut privat anomenat `studyPrograms` que serà un *array* (o col·lecció). Aquest *array* o col·lecció guardarà, com a mínim, un objecte de la classe `StudyProgram`.

En el segon exemple, una persona (`Person`) pot tenir diverses targetes de fidelització (`Card`) o cap, mentre que una targeta solament serà d'una persona. En aquest cas, la classe `Person` tindrà un atribut privat anomenat `cards` que emmagatzemarà objectes de tipus `Card` (pot ser un *array* buit). No obstant això, com que la navegabilitat és unidireccional, la classe `Card` no tindrà un atribut anomenat `holder`. De fet, es podria haver omès l'aparició d'aquest rol en l'associació. Entenem que en aquest context les targetes no tenen nom de titular.

1.6. Traduint a codi

Hem vist quatre tipus d'associacions des d'un punt de vista teòric, però com es codifiquen en la pràctica? Abans d'explicar com es codifiquen els diferents tipus d'associació que hem vist, cal tenir molt present el següent:

> La codificació de qualsevol associació ha de tenir-ne en compte les propietats, és a dir, la multiplicitat, la navegabilitat i els rols.

Vegem com afecta cada propietat a la codificació:

a) Navegabilitat. Com ja sabem, denota la consciència, per part d'una classe, de la pròpia relació. En termes d'implementació, la navegabilitat ens indica en quina de les classes hem de definir un atribut que sigui del tipus de l'altra classe. Dit d'una altra manera, la navegabilitat ens diu en quina classe ha d'haver-hi una referència a objectes de l'altra classe que participa en la relació. Així doncs, tenim dos casos:

- **Unidireccional**: la classe coneixedora (i.e. aquella de la qual surt la fletxa) serà l'única que tingui una referència a l'altra classe (i.e. aquella a la qual arriba la punta de la fletxa).

- **Bidireccional**: les dues classes de l'associació tindran una referència a l'altra classe.

b) Multiplicitat. A causa que la multiplicitat pot tenir diferents formats (p. ex. nombre exacte, rang de valors, etc.), la referència a l'altra classe dins de la classe coneixedora pot implementar-se de moltes maneres. No obstant això, podem diferenciar dos grans tipus de multiplicitat segons la manera en la qual es codifiquen:

- **Un objecte**. Quan la multiplicitat és `1` o `0..1`, llavors ens està dient que tenim un atribut en la classe coneixedora que és del tipus de l'altra classe.

- **Més d'un**. Quan la multiplicitat és diferent d'`1` o de `0..1` (per exemple, `*`, `1..5`, `8..12`, etc.), llavors l'atribut en la classe coneixedora ha de ser una col·lecció d'objectes de l'altra classe. Per *col·lecció* entenem qualsevol estructura que permeti emmagatzemar més d'un objecte, per exemple: un *array*, una pila, una cua, una llista, etc. Quan n'hem d'usar una o una altra? Depèn del problema que calgui resoldre.

c) Rol. Aquesta propietat ens ajuda a determinar dos aspectes de la implementació. D'una banda, ens dona a conèixer com el programador ha anomenat (o el dissenyador suggereix anomenar) l'atribut que fa referència a l'altra classe. D'altra banda, també ens indica el nivell d'accés d'aquest atribut. Com ja sabem, el nom del rol sol estar format pel modificador d'accés i el nom en si mateix, per exemple, `-person`, `-dogs`, etc.

Ara que sabem com afecten les propietats de l'associació a la seva codificació, vegem com afecta el tipus d'associació.

1.6.1. Associacions binàries, reflexives i d'agregació

Les associacions binàries, reflexives i d'agregació són idèntiques pel que fa a codificació. En el cas de les binàries i les reflexives és evident, ja que, com hem vist, aquestes últimes són un cas particular de les binàries en les quals les classes origen i destinació són la mateixa. Per la seva banda, les associacions binàries i d'agregació solament es diferencien des d'un punt de vista conceptual, ja que l'associació d'agregació, a diferència de la binària, afegeix la semàntica que en la relació hi ha una classe component que inclou a l'altra, però res més.

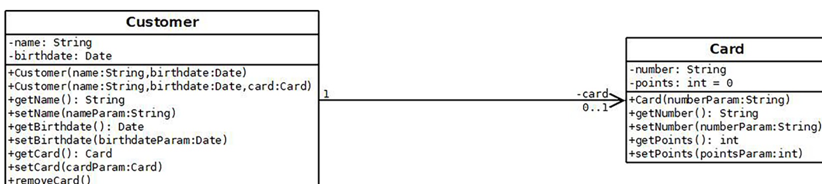
Com en els tres tipus d'associació, l'existència d'un objecte d'una classe és independent de l'existència d'un objecte de l'altra classe; la manera de codificar aquestes tres associacions és assignant un objecte ja existent a l'atribut que fa de rol en la classe coneixedora. És a dir, la classe coneixedora no instancia un objecte de l'altra classe, sinó que el rep i l'assigna a l'atribut que fa de rol.

Vegem-ne diversos exemples per entendre-ho millor:

> **Aclariment**
>
> Els exemples següents serveixen tant si l'associació és binària com reflexiva o d'agregació.

Exemple 8 (codificació) – Associació binària o agregació unidireccional 1-1

Atesa la relació següent entre client (`Customer`) i targeta de fidelització (`Card`).



Customer

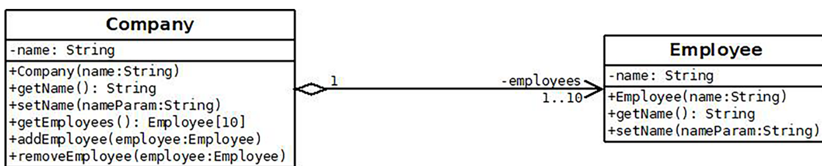
```
class Customer{
     private String title;
     private Date birthdate;
     private Car card;

     public Customer(String name, Date birthdate){
          setName(name);
          setYearBirth(yearBirth);
          card = null;
     }

     public Customer(String name, Date birthdate, Card card){
          setName(name);
          setBirthdate(birthdate);
          setCard(card);
     }

     public setCard(Card cardParam){
          if(cardParam != null){
               card = cardParam;
          }
     }
     public removeCard(){
          card = null;
     }
     …
}
```

Card

```
class Card{
     private String number;
     private int points; //No inclou un objecte de tipus Customer

     public Card(String numberParam){
          number = numberParam;
          points = 0;
     }
     …
}
```

Exemple 9 (codificació) – Associació binària o agregació unidireccional 1-N

Atesa la relació següent entre companyia (`Company`) i empleat (`Employee`).



Company

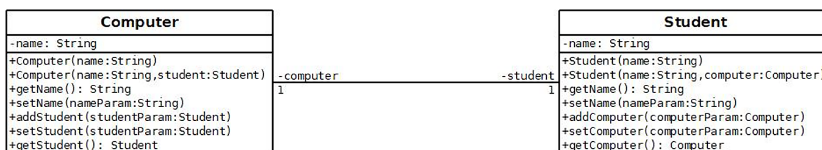
```
class Company{
     private String title;
     private Employee[10] employees;


     public Company(String name){
          setName(name);
          employees = new Employee[10];
     }

     public addEmployee(Employee employee){
          //Per a simplificar, suposem que els arrays tenen un mètode anomenat
          //contains que diu si hi ha l'objecte
          if(employee!= null && !employees.contains(employee)){
               employees.add(employee); //simplificació: suposem que els
     //arrays tenen el mètode "add" que afegeix l'objecte en la primera posició lliure
          }
     }

     …
}
```

Employee

```
class Employee{
     private String name; // No inclou un objecte de tipus Company

     public Employee(String name){
          setName(name);
     }
     …
}
```

Exemple 10 (codificació) – Associació binària o agregació bidireccional 1-1

Atesa la relació següent en la qual es dona cada estudiant (`Student`) un ordinador (`Computer`) a principi de curs.



Computer

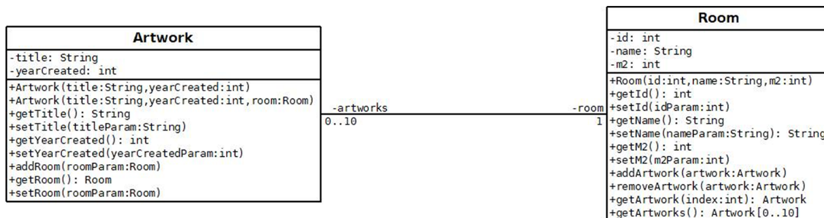
```
class Computer{
     private String name;
     private Student student;

     public Computer(String name, Student student){
          setName(name);
          addStudent(student);
     }

     public addStudent(Student studentParam){
          if(studentParam != null){
              //si student ja té ordinador, a aquest ordinador hi hem de
              //treure la referència a l'student
              if(studentParam.getComputer() != null){
                   studentParam.getComputer().setStudent(null);
              }
              setStudent(studentParam);//assignem student a aquest objecte Computer
              studentParam.setComputer(this); //assignem a student aquest objecte
                                   //Computer  (this)
          }
     }
     public setStudent(Student studentParam){
          student = studentParam;
     }

     public Student getStudent(){
          return student;
     }
     …
}
```

Student

```
class Student{
     private String title;
     private Computer computer;

     public Computer(String name, Student student){
          setName(name);
          addComputer(computer);
     }

     public addComputer(Computer computerParam){
          //Raonament idèntic que l'addStudent de la classe Computer
     }
     public setComputer(Computer computerParam) {
          computer = computerParam;
     }

     public Computer getComputer(){
          return computer;
     }
     …
}
```

Exemple 11 (codificació) – Associació binària o agregació bidireccional 1-N

Atesa la relació següent entre obres d'art (`Artwork`) i sales (`Room`) on s'exposen.



Artwork

```
class Artwork{
     private String title;
     private int yearCreated;
     private Room room;

     public Artwork(String title, int yearCreated){
          setTitle(title);
          setYearCreated(yearCreated);
     }

     public addRoom(Room roomParam){
          if(roomParam != null){
               if(!roomParam.getArtworks().contains(this)){
                    if(room != null){
                         room.removeArtwork(this);
                    }
                    setRoom(roomParam);
                    room.getArtworks().add(this);//simplificació:
                    //suposem que els arrays tenen el mètode "add" que
                    //afegeix l'objecte en la primera posició lliure

               }
          }
     }
     public Room getRoom(){
          return room;
     }

     public setRoom(Room roomParam){
          room = roomParam;
     }

     …
}
```

Room

```
class Room{
     private int id, m2;
     private String name;
     private Artwork[] artworks;

     public Room(int id, String name, int m2){
          setId(id);
          setName(name);
          setM2(m2);
          artworks = new Artwork[10];
     }

     public addArtwork(Artwork artwork){
          if(artwork != null && !artworks.contains(artwork)){
               if(artwork.getRoom()!=null){
                    artwork.getRoom().removeArtwork(artwork);
               }
               artwork.setRoom(this);
               artworks.add(artwork); //simplificació: suposem que els arrays
               //tenen el mètode "add" que afegeix l'objecte en la primera
               //posició lliure
          }
     }
     public removeArtwork(Artwork artwork){
          artworks.remove(artwork);
          //simplificació: suposem que els arrays tenen un
          //mètode "remove" que cerca l'objecte en l'array i l'esborra
     }

     public Artworks[] getArtworks(){
          return artworks;
     }
     …
}
```

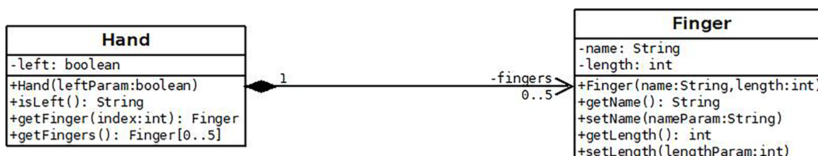
1.6.2. Associació de composició

Com ja sabem, a diferència dels tres tipus d'associació anteriors, la composició afegeix una restricció forta: l'existència d'un objecte de la classe component depèn de l'existència de la classe composta (o contenidora). A més, l'objecte de la classe component solament pot relacionar-se amb una classe composta mitjançant una associació de composició.

Per tot l'anterior, l'associació de composició es codifica instanciant l'objecte (o objectes) de la classe component dins de l'objecte de la classe composta. Aquesta instanciació normalment es fa dins del constructor de la classe composta. Aquesta manera de codificar l'associació de composició també implica que l'objecte de la classe composta sempre coneix l'objecte de la classe component.

Exemple 12 (codificació) – Associació de composició

Atesa la relació següent entre una mà (`Hand`) i els seus dits (`Finger`).



Hand

```
class Hand{
    private boolean left;
    private Finger[] fingers;

    public Hand(boolean leftParam){
        left = leftParam;
        fingers = new Finger[5];
        fingers[0] = new Finger("Thumb",2); //creem l'objecte del dit gros
        fingers[1] = new Finger("Index",5); //creem l'objecte del dit index
        fingers[2] = new Finger("Middle",8); //creem l'objecte del dit del cor
        fingers[3] = new Finger("Ring",6); //creem l'objecte del dit anular
        fingers[4] = new Finger("Little",4); //creem l'objecte del dit petit
    }
    …
}
```

Finger

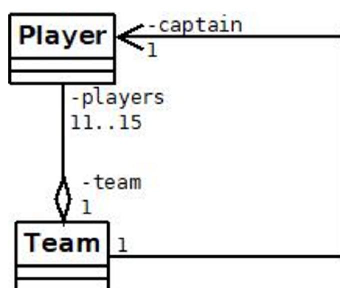
```
class Finger{
        private String name; //En aquest cas, no inclou un objecte de tipus Hand
        private int length;

        public Finger(String name, length){
                setName(name);
                setLength(length);
        }
        …
}
```

1.7. Dos objectes poden estar relacionats entre ells més d'una vegada?

La resposta és «sí». Dos objectes poden relacionar-se entre ells per mitjà de diferents associacions i, a més, aquestes associacions no han de ser necessàriament del mateix tipus. Vegem-ne un exemple:

Exemple 13 (dos objectes relacionats entre ells més d'una vegada) – Capità i jugador

Un equip, per exemple de bàsquet, té jugadors. Normalment, un dels jugadors de la plantilla és el capità de l'equip. Així doncs, l'objecte d'aquest jugador, a més de relacionar-se amb l'equip com a jugador, també es relaciona amb l'equip com a capità.



2. Classe associativa

2.1. Explicació

Aquest tipus de classe té sentit en el disseny del nostre programari i, per tant, pot aparèixer en un diagrama de classes UML. No obstant això, en la majoria de llenguatges de programació no hi ha una sintaxi especial que ens permeti definir una classe associativa.

Les classes associatives apareixen en les associacions (binàries, reflexives o d'agregació) en les quals necessitem guardar informació específica de la relació o associació (d'aquí el seu nom). A més, afegeix semàntica a la relació en forma de dues restriccions:

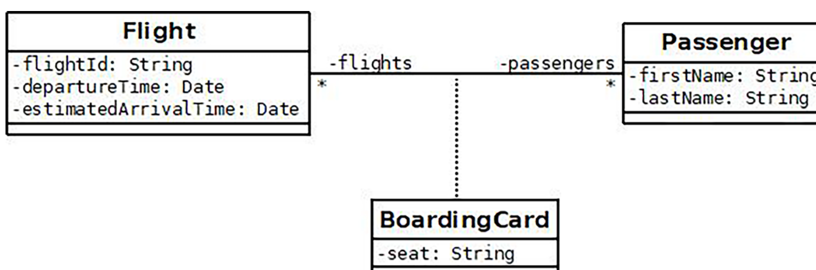
1) Donada una associació entre una classe `A` i una classe `B` en la qual hi ha una classe associativa `C`, **solament pot haver-hi un objecte de la classe `C` (classe associativa) per cada combinació d'objecte `A` i objecte `B` que estiguin relacionats**.

2) L'objecte de la classe associativa s'ha de destruir/eliminar quan un dels dos objectes que estan relacionats per l'associació desapareix (i.e., es destrueix) o quan s'elimina la relació entre tots dos objectes. Això és així perquè la classe associativa només té sentit quan els dos objectes de l'associació existeixen i estan relacionats.

Les classes associatives es representen amb una línia discontínua que va de la classe associativa a la línia contínua que relaciona l'associació.

Exemple 14 (classe associativa) – La targeta d'embarcament

Imaginem la relació en la qual un passatger (`Passenger`) té un seient assignat en un vol (`Flight`). En aquest cas, el seient no és ni del passatger ni del vol en si mateixos, sinó de la relació entre el passatger i el vol.



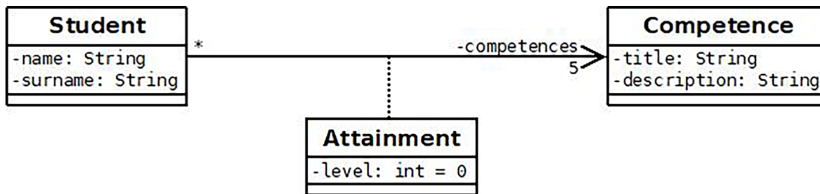
Com veiem, hem hagut de crear una classe anomenada `BoardingCard` que es relaciona amb l'associació. Aquesta nova classe incorpora informació, en aquest cas el seient (`seat`), que no és ni del vol ni del passatger pròpiament dit, sinó de tots dos. A més, es compleix la restricció que imposa la classe associativa que solament pot haver-hi un ob-

jecte `BoardingCard` per a un passatger concret en un vol concret. Un mateix passatger té dos seients en un vol? No.

Vegem un altre exemple:

Exemple 15 (classe associativa) – Nivell competencial dels estudiants

Imaginem que volem guardar el nivell d'assoliment que té cada estudiant (`Student`) en una competència transversal (`Competence`), com pot ser la comunicació oral o escrita, el treball en grup, etc. Un estudiant serà avaluat en cinc competències.



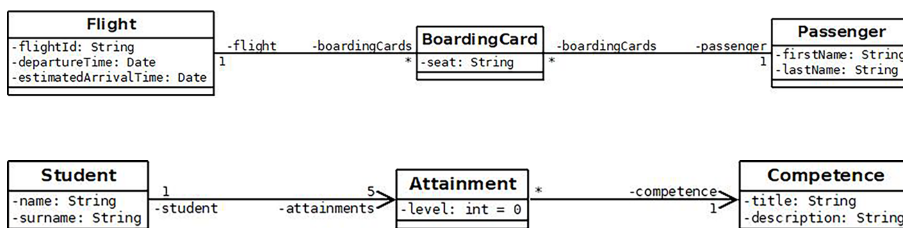
Com veiem, hem hagut de crear una classe anomenada `Attainment` que es relaciona amb l'associació. Aquesta nova classe incorpora informació, en aquest cas el nivell d'assoliment (`level`), que solament té sentit per al binomi estudiant-competència. Igual que en l'exemple anterior, un estudiant concret per a una competència concreta solament pot tenir un nivell d'assoliment, no més.

2.2. Traduint a codi

Com ja hem comentat, els llenguatges de programació, en general, no proporcionen un mecanisme directe per a implementar una classe associativa. Per aquest motiu, el més habitual és traduir o modificar l'associació en la qual participa la classe associativa per dues associacions binàries que van de les classes de l'associació binària original a la classe associativa.

Així mateix, per a mantenir la restricció que solament pot haver-hi un objecte de la classe associativa per cada combinació d'objectes de l'associació, és necessària una multiplicitat 1 en el costat de les classes no associatives. La multiplicitat en la classe associativa és la mateixa que hi havia en l'associació original. D'igual manera, la navegabilitat original també s'ha de respectar.

Vegem com es modifiquen els exemples anteriors per a poder-los codificar amb un llenguatge de programació:



Amb aquestes modificacions, ja podem codificar les classes com hem comentat per a les associacions binàries.

3. Classe parametritzada

3.1. Explicació

Una classe parametritzada, també coneguda com a *classe genèrica* (en Java i C#) o *template* (en C++), és aquella en la qual definim atributs de tipus genèric. Gràcies a això, la classe parametritzada encapsula operacions la funcionalitat de les quals no depèn d'un tipus d'objecte determinat. L'ús més comú d'aquest tipus de classe és el de crear classes que modelen el comportament d'una col·lecció, com per exemple: una pila, una cua, una llista, un arbre, una taula de *hash*, etc.

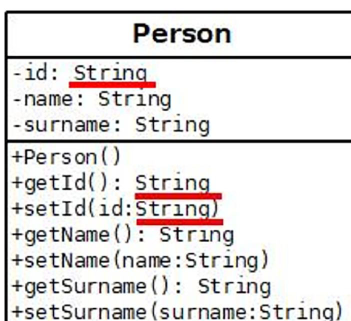
Les col·leccions tenen operacions o mètodes que es duen a terme de la mateixa manera, independentment del tipus d'objecte que emmagatzemen. Per exemple, si volem afegir un element a una pila, és igual si guardem un objecte de tipus `Person` o de tipus `Dog`, la manera de guardar l'objecte en la pila és la mateixa: posant-lo damunt de la col·lecció (i.e., en primer lloc; *last-in*, *first-out* - LIFO).

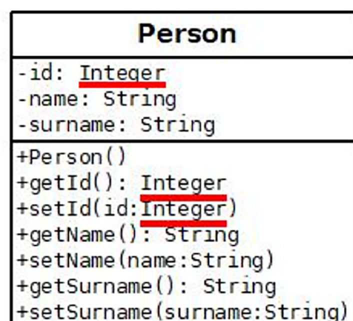
Llenguatges com Java o C# tenen en la seva API (és a dir, en el propi llenguatge) una gran quantitat de classes parametritzades que modelen col·leccions. Per exemple, Java proporciona les classes `ArrayList`, `LinkedList`, `HashMap`, `PriorityQueue`, `Stack`, entre d'altres. Per la seva banda, C# té `List`, `Dictionary`, `Queue`, `Stack`, etc.

Vegem un exemple en què ens pot ser útil fer una classe parametritzada:

Exemple 16 (classe parametritzada) – Una classe `Person` per a diversos programes

Imagina que estem programant en diversos projectes alhora i en tots dos ens demanen que fem una classe `Person` a partir dels diagrames de classes següents.



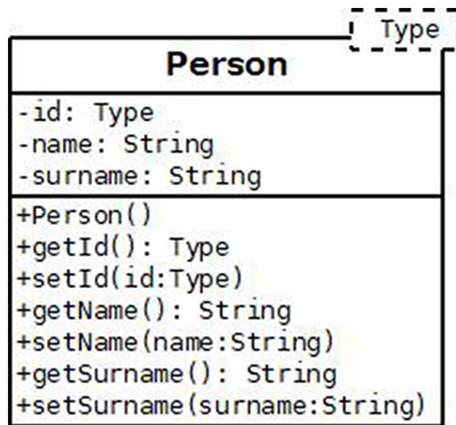


Project 1

Project 2

Si ens hi fixem, veiem que les classes són pràcticament idèntiques, l'única cosa que canvia és el tipus de l'atribut `id`, el qual és un `String` en el primer projecte i un enter (`Integer`) en el segon projecte.

Així doncs, en comptes de codificar dues vegades la classe `Person` amb el petit canvi del tipus de l'atribut `id`, decidim fer una classe parametritzada.



Com podem veure, ara el tipus de l'atribut `id` és `Type`, que pot ser qualsevol classe. Així doncs, podem instanciar un objecte de la classe `Person` de les maneres següents (entre d'altres):

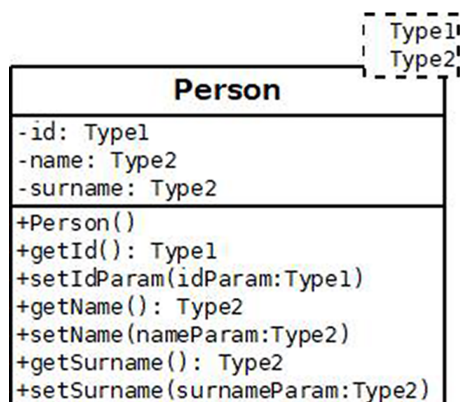
```
Person<String> personProject1 = new Person<String>();
Person<Integer> personProject2 = new Person<Integer>();
```

El tipus de l'atribut `id` serà un paràmetre de la mateixa classe. En el nostre exemple de codi, hem indicat el tipus mitjançant <> (pensa que el compilador substitueix `Type` per `String` o `Integer` o qualsevol classe que indiquem, segons sigui el cas).

En l'UML s'indica el nom que es dona al tipus genèric (l'hem anomenat `Type`, però podria haver estat un altre nom) mitjançant una petita caixa situada en la part superior dreta.

En l'exemple anterior, gràcies a l'ús d'una classe parametritzada, hem aconseguit codificar una sola vegada la classe `Person` abstraient-nos del tipus de l'atribut `id`. Ara, la classe `Person` és genèrica, és a dir, serveix per a qualsevol context en què solament l'atribut `id` canvia de tipus.

Una classe parametritzada pot rebre tants tipus com es necessitin. Vegem-ne un exemple en el qual hi ha dos tipus:



3.2. Traduint a codi

Cada llenguatge de programació té les seves particularitats i regles a l'hora de definir les classes parametritzades. Perquè pugueu entendre les classes parametritzades des d'un punt de vista de la codificació, en termes generals podríem dir que la seva codificació seria com segueix:

```
class Person<Type>{
     private Type id;
     private String name, surname;

     public Person>(){}

     public Type getId(){ return id;}

     public setId(Type idParam){ id = idParam;}
     …
}
```

Així mateix, no tots els llenguatges de programació suporten la definició i ús de classes parametritzades.

Resum

Associacions

- Una associació és la relació entre objectes o instàncies. Dos objectes poden estar relacionats entre ells per més d'una associació.

- Els tipus d'associació principals són els següents: binària, d'agregació, de composició i reflexiva.

- L'associació binària es dona entre dues instàncies en què cada una existeix de manera independent a l'existència de l'altra.

- L'associació d'agregació és similar a la binària, però incorpora la semàntica que un dels dos objectes és component de l'altra (anomenada *composta*).

- L'associació reflexiva és un cas particular de l'associació binària en la qual un objecte d'una classe es relaciona amb un altre objecte de la mateixa classe.

- Tant l'associació binària com la d'agregació i la reflexiva es codifiquen de la mateixa manera.

- L'associació de composició és com la d'agregació, però l'existència de l'objecte component depèn de l'existència de l'objecte compost. A més, l'objecte component solament pot relacionar-se mitjançant composició amb un únic objecte compost. No obstant això, pot relacionar-se amb altres objectes mitjançant altres tipus d'associació.

- Tota associació té tres propietats: multiplicitat, navegabilitat i rol.

- La multiplicitat està formada pel nombre mínim i màxim d'objectes que participen d'una classe en la relació. El nombre mínim s'anomena *opcionalitat* o *participació*, mentre que el nombre màxim es denomina *cardinalitat*. Cada associació té dues multiplicitats, una per cada classe que participa en la relació.

- La navegabilitat indica si un objecte d'una classe coneix l'existència d'un objecte de l'altra classe. Si solament un objecte de l'associació té coneixement de la relació, llavors la navegabilitat és unidireccional. En cas contrari, és bidireccional.

- El rol és una etiqueta que serveix per a aportar informació contextual a l'associació. A més, ajuda a l'hora de codificar perquè suggereix un nom

per a l'atribut d'una classe que fa de referència als objectes de l'altra classe de la relació.

Classe associativa

- Les classes associatives apareixen en les associacions (binàries, reflexives o d'agregació) en què necessitem guardar informació específica de la relació o associació.

- Solament pot haver-hi un objecte de la classe associativa per cada combinació d'objecte `A` i objecte `B` que estiguin relacionats mitjançant l'associació.

- Quan un dels dos objectes que estan relacionats per l'associació desapareix (i.e., es destrueix) o quan s'elimina l'associació entre tots dos objectes, llavors l'objecte de la classe associativa també ha de destruir-se.

- La majoria dels llenguatges de programació no permeten crear classes associatives. Per aquest motiu, hem de modificar o transformar la classe associativa per una classe normal, i convertir l'associació en la qual participa la classe associativa en dues associacions binàries que van de les classes de l'associació binària original a la classe associativa.

Vídeo d'interès

Pots veure un resum dels tipus d'associació i de la classe associativa en el vídeo «Relació entre objectes» que trobaràs a l'aula de l'assignatura.

Classe parametritzada

- Una classe parametritzada ens permet abstreure'ns del tipus de tots o alguns dels atributs de la classe encapsulant operacions la funcionalitat de les quals no depèn del tipus d'objecte que sigui l'atribut d'aquesta classe.

- Gràcies a una classe parametritzada, generalitzem l'ús de la classe a diferents contextos i, al mateix temps, estalviem en hores de codificació.

- L'ús més freqüent d'una classe parametritzada és en la implementació de col·leccions: pila, cua, llista, etc.

- La classe parametritzada també és coneguda com a *classe genèrica* o *template*.

Bibliografia

Booch, G.; Maksimchuk, R.; Engle, M.; Young, B. J.; Conallen, J.; Houston, K. i altres (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.

Microsoft (s. f.). «C# Guide» [en línia]. Microsoft. <https://docs.microsoft.com/en-us/dotnet/csharp/>

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.

Seidl, M.; Scholz, M.; Huemer, C.; Kappel, G. (2015). *UML @ Classroom: An Introduction to Object-Oriented Modeling*. Heidelberg: Springer.

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

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