CFGS DESENVOLUPAMENT D'APLICACIONS MULTIPLATAFORMA (DAM)
MP03 – PROGRAMACIÓ
UF4: PROGRAMACIÓ ORIENTADA A OBJECTES. FONAMENTS

INTRODUCCIÓ A LA PROGRAMACIÓ ORIENTADA A OBJECTES

1. Introducció

La programació orientada a objectes (POO) és el que es coneix com un ***paradigma*** o **model** de programació. Això significa que no és cap llenguatge de programació, sinó una forma de programar. La POO s'ha convertit en una de les formes de programar més populars. Però primer, veiguem perquè va sorgir este concepte.

1.1 L'origen de la POO

La POO sorgix com un intent per dominar la complexitat que, de forma innata, posseeix el programari. Tradicionalment, la forma d'enfrontar-nos a esta complexitat ha sigut empleant el que anomenem **programació estructurada**, que consisteix en descompondre el problema que volem sol·lucionar en subproblemes i més subproblemes fins arribar a accions molt simples i fàcils de codificar.

La POO és un altra forma de descompondre els problemes. Este nou mètode de descomposició és la ***descomposició en objectes***; **anem a fixar-nos** no en el que s'ha de fer en el problema, sinó **en quin és l'escenari real** del mateix, **i anem a intentar simular este escenari en el nostre programa**.

Els primers conceptes de la programació orientada a objectes tenen origen en **Simula 67**, un llenguatge dissenyat per fer simulacions, creat per Ole-Johan Dahl i Kristen Nygaard, del *Centre de Còmput Noruec* a Oslo. En este centre es treballava en simulacions de naus. La idea va sorgir a l'agrupar els diversos tipus de naus en diverses classes d'objectes, sent responsable cada classe d'objectes de definir les seves pròpies dades i comportaments.

Van ser refinats més tard en **Smalltalk**, desenvolupat en Simula en Xerox PARC (la primera versió va ser escrita sobre **Basic**) però dissenyat per a ser un sistema completament dinàmic en el qual els objectes es podrien crear i modificar "sobre la marxa" (en temps d'execució) en lloc de tenir un sistema basat en programes estàtics.

La programació orientada a objectes es va anar convertint en l'estil de programació dominant a mitjans dels anys 80, en gran part a causa de la influència de **C++**, una extensió del llenguatge de programació **C**. El seu domini va ser consolidat gràcies a la popularitat de les *Interfícies Gràfiques d'Usuari* (GUI), per a les quals la programació orientada a objectes està particularment ben adaptada. En este cas, es parla també de programació dirigida per esdeveniments.

Les característiques d'orientació a objectes van ser agregades a molts llenguatges existents durant este temps, incloent **Ada**, **BASIC**, **Lisp** i **Pascal**, entre d'altres. L'addició d'estes característiques als llenguatges que no van ser dissenyats inicialment per a elles va conduir sovint a problemes de compatibilitat i en la capacitat de manteniment del codi.

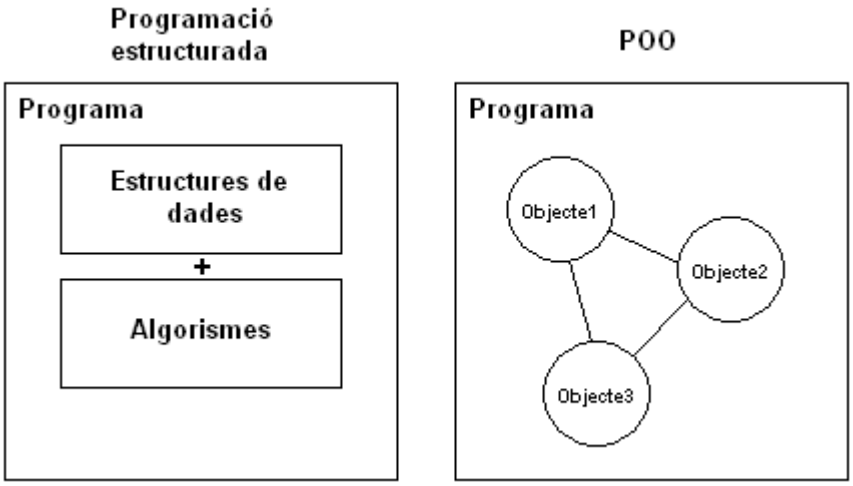
Els llenguatges orientats a objectes "purs", per la seva banda, no tenien les característiques de les quals molts programadors havien vingut a dependre. Per saltar este obstacle, es van fer moltes temptatives per crear nous llenguatges basats en mètodes orientats a objectes, però permetent algunes característiques imperatives de maneres "segures". El **Eiffel** de Bertrand Meyer va ser un primerenc i moderadament encertat llenguatge amb estos objectius, però ara ha estat essencialment reemplaçat per **Java**, en gran part a causa de l'aparició d'Internet i a la implementació de la màquina virtual de Java (**JVM**) en la majoria de navegadors. **PHP** en la seva versió 5 s'ha modificat; suporta una orientació completa a objectes, complint totes les característiques pròpies de l'orientació a objectes.

1.2 Diferències entre programació estructurada i POO

Els llenguatges de programació tradicionals no orientats a objectes, com **C**, **Pascal**, **BASIC**, o **Modula**, basen el seu funcionament en el concepte de *procediment* o *funció*. Una funció és simplement un conjunt d'instruccions que operen sobre uns arguments i produeixen uns resultats. D'esta manera, un programa estructurat no és més que una successió de crides a funcions, ja siguin estes del sistema operatiu, proporcionades pel propi llenguatge o desenvolupades pel propi usuari.

En el cas dels llenguatges orientats a objectes, com és el cas de **C++**, **Java** o **C#**, l'element bàsic no és la funció, sinó els **objectes**. Un objecte és la representació en el programa d'un *concepte*, i conté tota la informació necessària per abstraure-ho: dades que descriuen els seus atributs i operacions que poden realitzar-se sobre els mateixos descrits mitjançant mètodes.

El que caracteritza a la POO és que intenta portar al món del codi el mateix que ens trobem en el món real. Quan mirem al nostre voltant veiem coses, objectes, i cada *objecte* pertany a una *classe* diferent. És el que ens permet distingir a un cotxe d'un gos, per exemple, ja que pertanyen a classes diferents. Per tant, **la POO expressa un programa com un conjunt d'objectes que col·laboren entre ells per a realitzar tasques**.



Diferència entre programació estructurada i POO

Podem dir que un llenguatge de programació és orientat a objectes si presenta les característiques següents:

- Objectes i classes.
- Encapsulament.
- Herència i polimorfisme.

Estes són els conceptes que veurem al llarg d'este tema.

1.3 Un primer exemple

Si ens parem a pensar sobre cóm se'ns planteja un problema qualsevol en la realitat podrem veure que el que hi ha en la realitat són ***entitats*** (és a dir, *objectes*). Estes entitats posseeixen un conjunt de propietats o ***atributs***, i un conjunt de ***mètodes*** mitjançant els quals ens mostren el seu *comportament*. I no només això, també podem descobrir, a poc que ens fixem, tot un conjunt d'***interrelacions*** entre les entitats, guiades per l'intercanvi de ***missatges***; les entitats del problema responen a estos missatges mitjançant l'execució de certes *accions*.

El següent exemple, encara que paregui molt estrany, ens ajudarà a aclarir estos conceptes. Imaginem la següent situació:

Un diumenge per la tarde estic a casa veient la televisió, i de repent, ma mare sent un fort dolor de cap; com és natural, el primer que faig és tractar de trobar un caixeta d'aspirines.

El que s'acaba de descriure és una situació molt normal en el nostre dia a dia. Anem a vore-la en clau d'objectes: l'objecte fill ha rebut una missatge procedent de l'objecte mare. L'objecte fill respon al missatge mitjançant una acció: buscar aspirines. La mare no ha de dir-li al fill on ha de buscar les aspirines, és responsabilitat del fill resoldre el problema com ell consideri oportú. A l'objecte mare li basta amb haver emitit el missatge. Continuem amb la història:

El fill no troba aspirines en la farmaciola i decideix acudir a la farmàcia de guàrdia més propera per comprar aspirines. En la farmàcia és atès per la farmacèutica que li pregunta què desitja, i el fill respon: "una caixeta d'aspirines, per favor". La farmacèutica desapareix i retorna al poc temps amb la caixa d'aspirines en la mà. El fill paga l'import, s'acomiada i torna a sa casa. Allí li dóna el comprimit a sa mare, la qual en una estona comença a experimentar una notable milloria fins la completa desaparició del dolor de cap.

El fill, per a resoldre el problema, entra en relació en un nou objecte, la farmacèutica, qui respon al missatge del objecte fill amb la búsqueda de la caixeta d'aspirines. L'objecte farmacèutica és ara el responsable de la solució del problema. L'objecte farmacèutica llança un missatge a l'objecte fill sol·licitant el pagament de l'import, i l'objecte fill respon a tal esdeveniment amb l'acció de pagar.

Com hem pogut veure, en esta situació ens trobem amb objectes que es diferencien dels altres per un conjunt de característiques o propietats, i per un conjunt d'accions que realitzaven en resposta a uns esdeveniments que s'originaven en altres objectes o en el entorn.

També podem donar-nos compte que, encara que els objectes tenen propietats distintes, com el color del cabell, l'edat, el pes, etc., tots tenen un conjunt d'atributs comuns (ja que tots són *persones*). **A este patró d'objectes l'anomenarem *classe***.

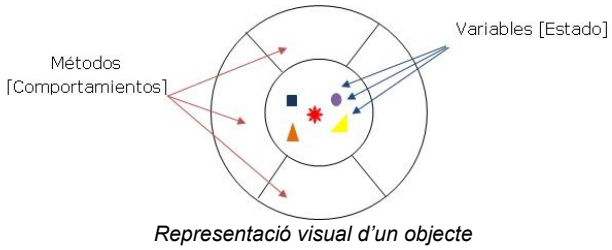
2. Objectes

2.1 Concepte d'objecte

Un **objecte** és una *entitat* que es correspon amb els objectes reals del món que ens rodeja i representa una ***instància* d'una *classe*** del nostre programa. Els objectes posseeixen:

- Estat (*atributs o variables*): conté informació que pot emmagatzemar l'objecte. Pot variar respecte al temps.
- Comportament (*mètodes*): conjunt d'accions que pot realitzar.
- Identitat: propietat que fa d'un objecte distingible de la resta d'objectes.

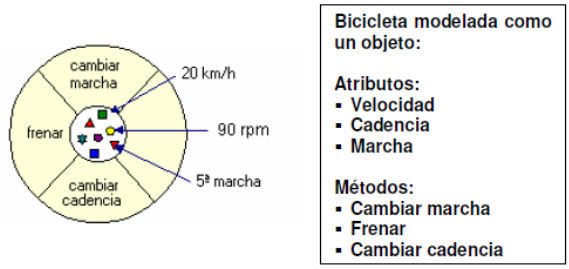
L'objecte obté la seua identitat en el moment de la seua creació, la qual no canviarà al llarg de tota la seua vida. La vida d'un objecte finalitza quan es destrueix. La següent figura mostra una representació visual d'un objecte:



Representació visual d'un objecte

Els *atributs* de l'objecte (estat) i el que l'objecte pot realitzar (comportament) estan expresats per les variables i mètodes que composen l'objecte, respectivament. Per exemple, un objecte que modela una bicicleta del món real tindria unes variables que indicarien l'estat actual de la bicicleta: la seua velocitat és de 20km/h, la seua cadència de pedaleig 90rpm, i la seua marxa actual és la 5ª. Estes variables es coneixen formalment com ***variables d'instància*** o *variables membre* perquè contenen l'estat d'un objecte bicicleta particular, i en programació orientada a objectes, un **objecte particular** es denomina una ***instància***.

A més d'estes *variables*, l'objecte bicicleta podria tindre *mètodes* per a frenar, canviar la cadència de pedaleig, i canviar la marxa. Estos mètodes es denominen formalment ***mètodes d'instància*** o *mètodes membre*, ja que canvien l'estat d'una instància u objecte bicicleta particular. La següent figura mostra una bicicleta modelada com un objecte:



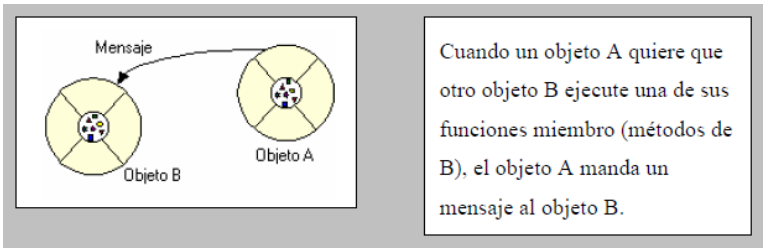
Objecte bicicleta

La figura anterior mostra les variables objecte en el nucli o centre de l'objecte, i els mètodes rodejant al nucli i protegint-lo d'altres objectes del programa. El fet d'empaquetar o protegir les variables membre amb els mètodes membre es denomina **encapsulament**.

Este exemple mostra la representació ideal d'un objecte. No obstant això, en ocasions, és possible que un objecte tingui que exposar algunes de les seues variables membre, o protegir altres dels seus propis mètodes o funcions membre. Per exemple, **Java** permet establir 4 tipus de protecció de les variables i mètodes per casos com este. Els nivells de protecció (*public, private, protected, static*, etc.) determinen quins objectes i classes poden accedir a quines variables i a quins mètodes.

2.2 Concepte de missatge

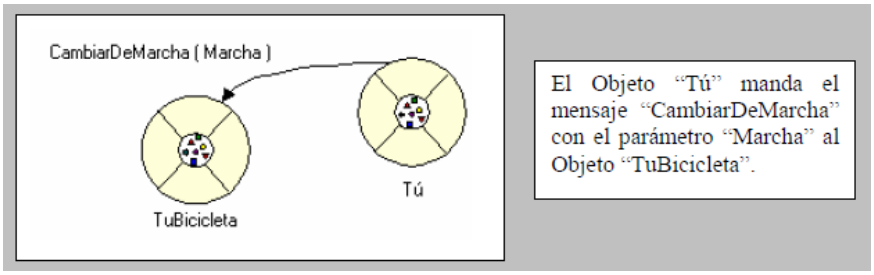
Normalment, un únic objecte per sí mateix no és molt útil. En general, un objecte apareix com un component més d'un programa o una aplicació que conté altres molt objectes. Els objectes d'un programa interactuen i es comuniquen entre ells mitjançant els missatges. Quan un objecte A vol que un altre objecte B execute alguna de les seues funcions membre (mètodes de B), l'objecte A envia un missatge a l'objecte B.



En ocasions, l'objecte que rep el missatge necessita més informació per saber exactament el que té que fer; per exemple, quan es desitja canviar la marxa d'una bicicleta, s'ha d'indicar la marxa a la que es vol canviar. Esta informació es passa junt amb el missatge en forma de **paràmetre**.

La següent imatge mostra les tres parts que componen un missatge:

- L'objecte al qual s'envia el missatge (Bicicleta).
- El mètode o funció membre que ha d'executar (CanviarDeMarxa).
- Els paràmetres que necessita este mètode (Marxa).



Estes tres parts del missatge (objecte destinatari, mètode i paràmetres) són suficient informació per a que l'objecte que rep el missatge executi el mètode o la funció membre solicitada.

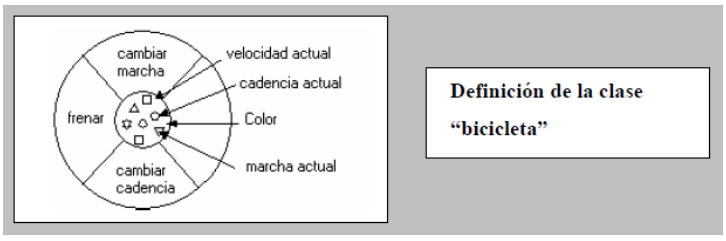
3. Classes

Normalment, en el món existeixen diversos objectes d'un mateix tipus, o com direm de seguida, d'una mateixa *classe*. Per exemple, la meua bicicleta és una de les moltes bicicletes que existeixen en el món. Utilitzant la terminologia de la programació orientada a objectes, direm que la meua bicicleta és una **instància** de la **classe** d'objecte coneguda com *bicicletes*.

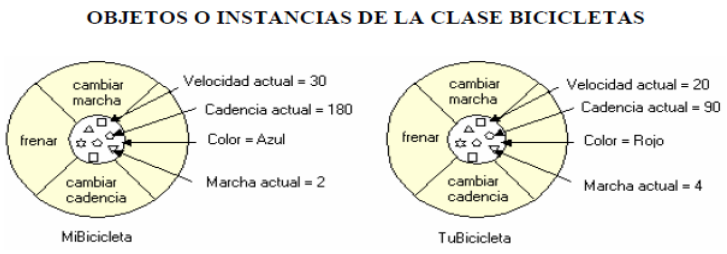
Totes les bicicletes tenen alguns atributs (color, marxa, cadència, rodes, etc.) i alguns mètodes (canviar marxa, frenar, etc.) en comú. No obstant això, l'estat particular de cada bicicleta és independent de l'estat de les altres bicicletes. La particularització d'estos atributs pot ser diferent. És a dir, una bicicleta podrà ser blava, un altra verda, però les dues tenen en comú el fet de tindre una variable "color". D'esta manera podem definir una plantilla d'atributs i mètodes per a totes les bicicletes. Les plantilles per a crear objectes són denominades **classes**.

Per tant, **una classe és una definició o plantilla que defineix els atributs i mètodes que són comuns per a tots els objectes d'un cert tipus**, i permet la creació d'objectes d'este mateix tipus.

En el nostre exemple, la classe bicicleta es representaria de la següent forma:



Després d'haver creat la classe bicicleta, podem crear qualsevol número d'objectes bicicleta a partir de la classe. Quan creem una instància d'una classe, el sistema reserva suficient memòria per a l'objecte amb tots els atributs membre. Cada objecte té la seua pròpia còpia de les variables membre definides en la classe.



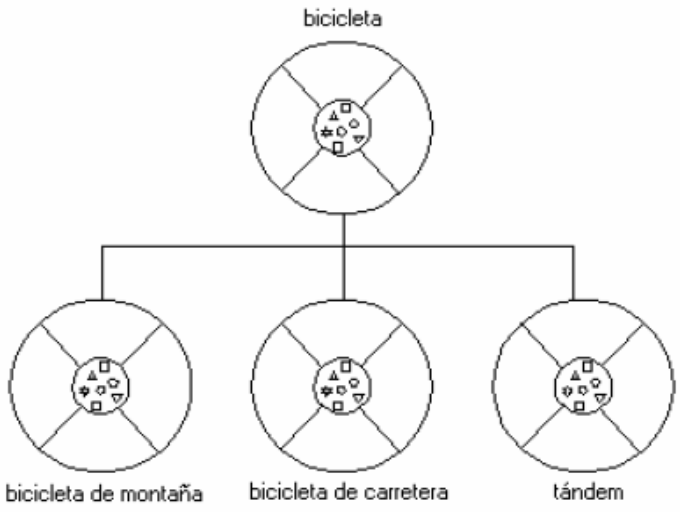
4. Herència

4.1 Concepte d'herència

Una vegada hem vist el concepte d'objecte i el de classe, ens trobem en condicions d'introduir un altra de les característiques bàsiques de la programació orientada a objectes: el concepte d'**herència**.

El mecanisme d'herència permet definir noves classes a partir d'altres ja existents. Les classes que *deriven* d'altres hereten automàticament tot el seu comportament, però a més poden introduir característiques particulars pròpies que les diferencien.

Per exemple, les bicicletes de muntanya, les de carretera i els tàndem són totes, en definitiva, bicicletes. En termes de programació orientada a objectes, són *subclasses* o *classes derivades* de la classe bicicleta. Anàlogament, la classe bicicleta és la *classe base* o *superclasse* de les bicicletes de muntanya, les de carretera i els tàndems.



L'herència proporciona els següents avantatges:

- Les classes derivades o subclasses proporcionen **comportaments especialitzats** a partir d'elements comuns que hereten de la classe base. Mitjançant el mecanisme d'herència els programadors poden reutilitzar el codi de la superclasse tantes vegades como sigui necessari.

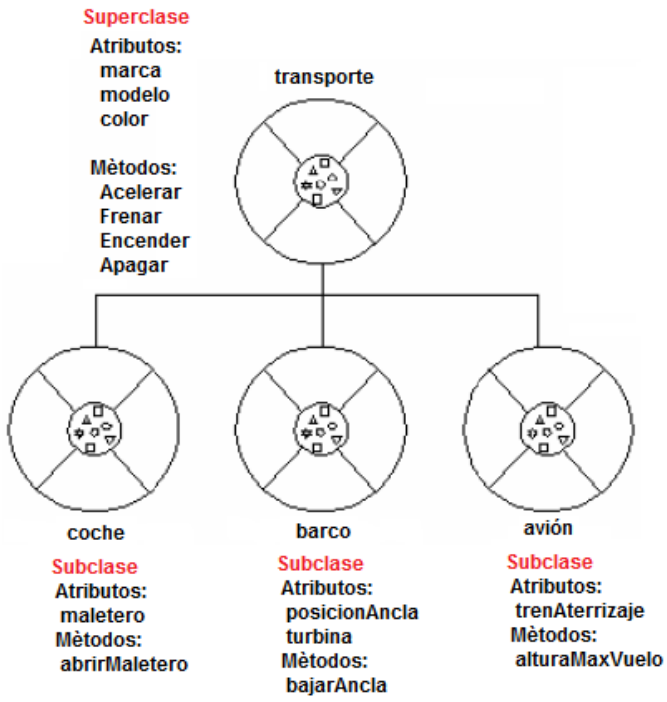
- Els programadors poden implementar les crides a les superclasses abstractes, que defineixen comportaments genèrics. Les **classes abstractes** defineixen i implementen parcialment comportaments, però gran part d'estos comportaments no es defineixen ni s'implementen totalment. D'esta manera, altres programadors poden fer ús d'estes superclasses detallant estos comportaments amb subclasses especialitzades. El propòsit d'una **classe abstracta** és servir de model de base per a la creació d'altres classes derivades, però on la seua implementació depengui de les característiques particulars de cada una d'elles. Un exemple de la classe abstracta podria ser en el nostre cas la classe *vehicle*. Esta classe seria una classe base genèrica a partir de la qual podriem anar creant tot tipus de classes derivades.

4.1 Tipus d'herència

L'herència és classifica segons el mode d'accés als membres de la classe base:

- Public
- Protected
- Private

A més, l'herència també pot ser **simple** (cada classe té una sola superclasse) o **múltiple** (cada classe pot tindre associada varies superclasses).



Exemple d'herència simple

Exemple d'herència múltiple

5. Polimorfisme

5. 1 Concepte de polimorfisme

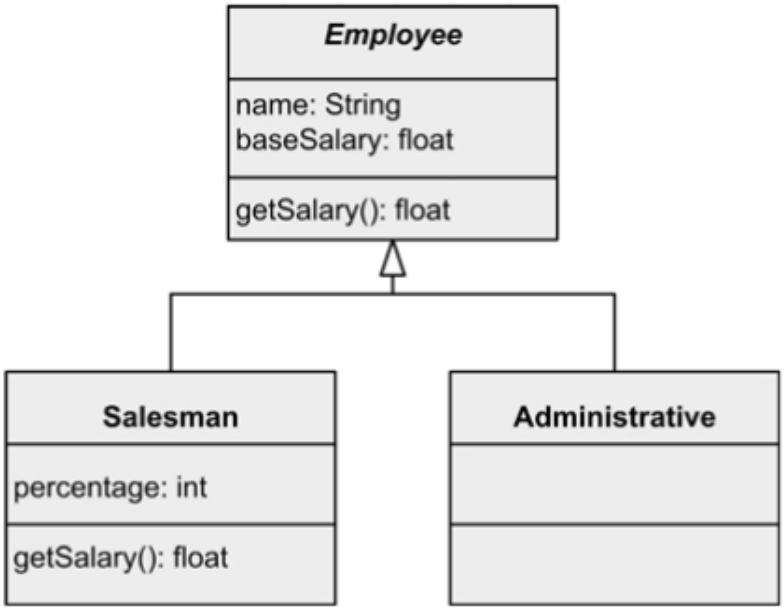
El mot polimorfisme prové dels termes grecs poli ('diversos') i morfos ('forma'), els quals defineixen perfectament este concepte.

Polimorfisme és la **propietat d'ocultar l'estructura interna d'una jerarquia de classes implementant un conjunt de mètodes de manera independent i diferenciada en cada classe**.

Este concepte apareix quan definim un mètode en una classe de la jerarquia (generalment la superclasse) i el reescrivim en, com a mínim, alguna de les classes que formen la jerarquia.

A l'exemple següent veiem que el polimorfisme sorgeix de la necessitat de modificar el comportament d'una operació per a una classe concreta (o un conjunt de classes que formen una jerarquia nova).

El



polimorfisme permet que es decideixi en temps d'execució i de manera automàtica quin dels mètodes cal executar, és a dir, el **mètode heretat** o, en cas que existeixi, el **mètode sobreescrit**.

L'ús del polimorfisme permet separar el que s'ha de realitzar de la manera com es realitza.

Les **superclasses** (generalment abstractes) **declaren el que s'ha de fer** i les **subclasses implementen els mètodes segons el cas**, encara que de vegades la superclasse també implementa el comportament estàndard per tal que només les subclasses que necessiten un comportament diferent sobreescriguin el mètode.

El concepte de polimorfisme està fortament lligat al concepte d'herència, sense el qual no tindria sentit.

Encara que el concepte de polimorfisme i el de sobrecàrrega de mètodes puguin semblar similars, si no iguals, cal diferenciar-los clarament, ja que la sobrecàrrega consisteix a definir un mètode nou amb una signatura diferent (nombre i tipus de paràmetres) i el polimorfisme és la substitució d'un mètode per un altre en una subclasse mantenint la signatura original. Una altra diferència que podem trobar és que la sobrecàrrega es pot detectar en temps de compilació i, en canvi, el polimorfisme es resol en temps d'execució.

6. Annex

En les següents taules es mostren diferents llenguatges de programació ordenats cronològicament i segons el paradigma o finalitat amb la qual s'han creat.

Anys	Paradigma	Llenguatges
60 - 70	Estructurada	Fortran Basic Pascal Cobol Modula C ...
	Funcional	LISP ...
80 – actualitat...	**OO**	Smalltalk Delphi C++ Java C# Phyton ...

Taula cronològica amb els principals llenguatges de programació

Anys	Ús	Llenguatges
90 – actualitat...	Web	HTML Javascript XML PHP ASP.Net JSP Phyton ...
70 – actualidad...	Base de dades	SQL QBE ...
	Mobile	Java .Net C# Phyton ...