TEMA 6

PROGRAMACIÓ ORIENTADA A OBJECTES

Índex de continguts

1 INTRODUCCIÓ

La programació orientada a objectes també coneguda com POO és el paradigma de programació que més s'utilitza hui en dia i que permet modelar de manera més eficient la realitat des del punt de vista del programari.

La programació orientada a objectes intenta proporcionar un model de programació basat en objectes que contenen dades i procediments associats coneguts com a mètodes. Aquests objectes, que són instàncies de les classes, són tipus abstractes de dades que encapsulen (amaguen) tant les dades com les funcions per accedir-hi.

Un dels objectius de la programació orientada a objectes és reflectir la realitat, de forma que els elements d'un programa s'ajusten a elements de la vida quotidiana. Per exemple, suposem que volem realitzar una aplicació per un taller de vehicles. En un programa estructurat definiríem funcions independentment de les dades, tal que per un costat crearíem el codi de la funció i posteriorment al programa principal s'establiria una variable o variables sobre les que aplicaríem canvis, podria ser un array d'una estructura a la que guardaríem tipus de vehicle, matrícula, color, etc. En programació orientada a objectes creem un objecte que simula un cotxe amb les seues característiques generals i les funcions incloses, tal que en la funció principal de la nostra aplicació, crearem variables d'aquest objecte i usarem els mètodes aplicats a unes dades concretes a cada moment.

La programació orientada a objectes permet la creació de programari cada vegada més complexe a partir d'unitats o blocs de codi reutilitzables.

2 CARACTERÍSTIQUES PRINCIPALS

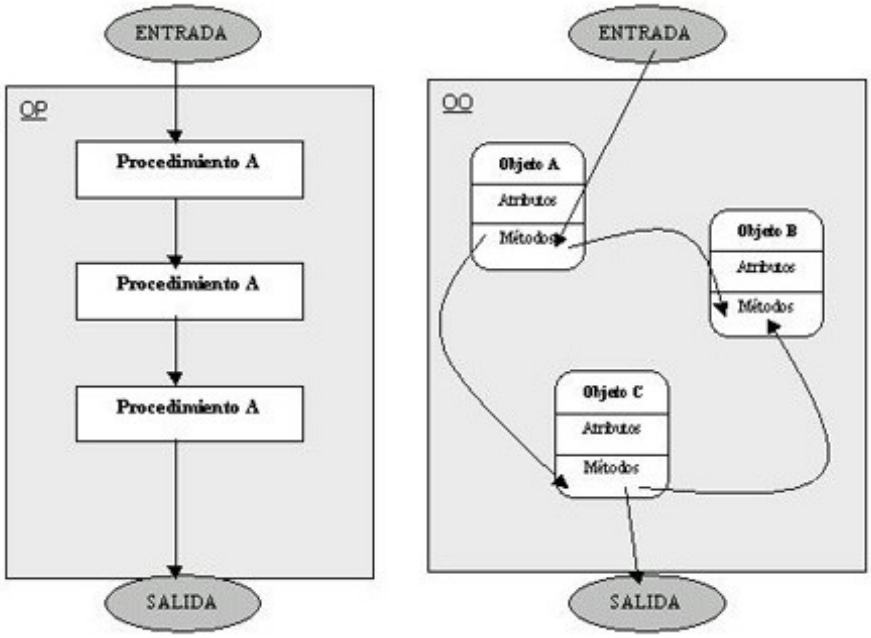
Les propietats principals o característiques fonamentals del paradigma de programació orientada a objectes són:

2.1 ABSTRACCIÓ

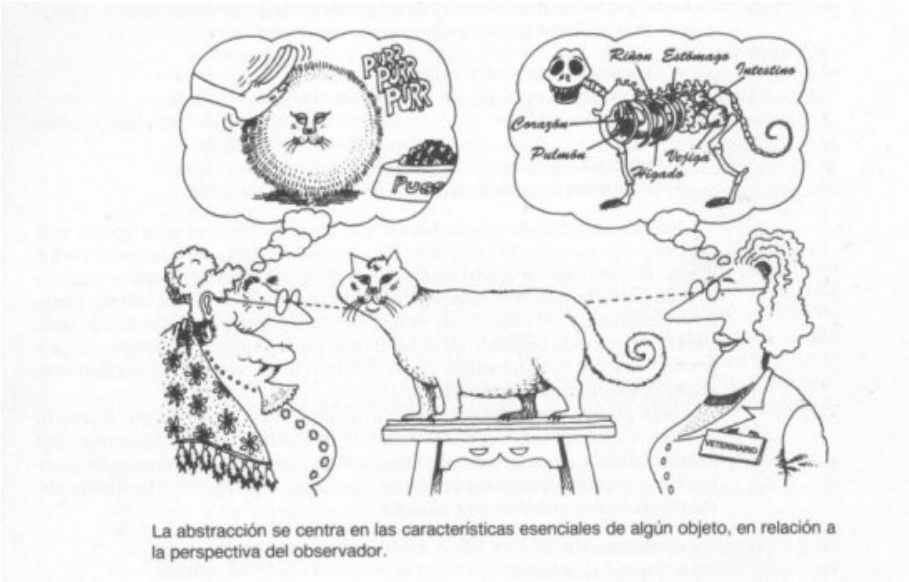
Abstraure's és aïllar mentalment, considerar separadament (un o diversos atributs o qualitats) d'una cosa. Per exemple abstraure els conceptes més importants d'un discurs, és a dir, quedar-se amb el important.

En programació orientada a objectes abstraure's, l'abstracció, seria mostrar únicament les característiques essencials, que fa l'objecte i per a que es crea, deixant de banda elements com la implementació.

L'abstracció permet representar les característiques essencials d'un objecte, deixant enrere aquelles que no tenen tanta importància. A més a més, es centra en l'objecte tal i com el coneixem en la vida real, de forma que ens centrem en allò que es capaç de fer però no en com ho fa. Bàsicament, definiríem l'abstracció com la forma de descriure una entitat del món real sense importar la complexitat que està present i el poder utilitzar aquesta en qualsevol aplicació.



Un altre exemple d'abstracció



La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.

2.2 ENCAPSULAMENT

L'encapsulament és, en altres paraules, l'ocultació d'informació, de forma que les dades internes d'un objecte estan ocultes al món exterior, només es coneix d'ell la seua essència, és a dir, què podem fer amb ell.

Per exemple, amb freqüència es desenvolupa codi font o projectes en els que utilitzem la classe out, més concretament el mètode println. Out es troba dins de l'espai de noms (namespace) System i no és més que una classe d'aquest paquet. Dit en altres paraules. Out és una classe i println() és un mètode de la classe Out.

El encapsulamiento oculta los detalles de la implementación de un objeto.

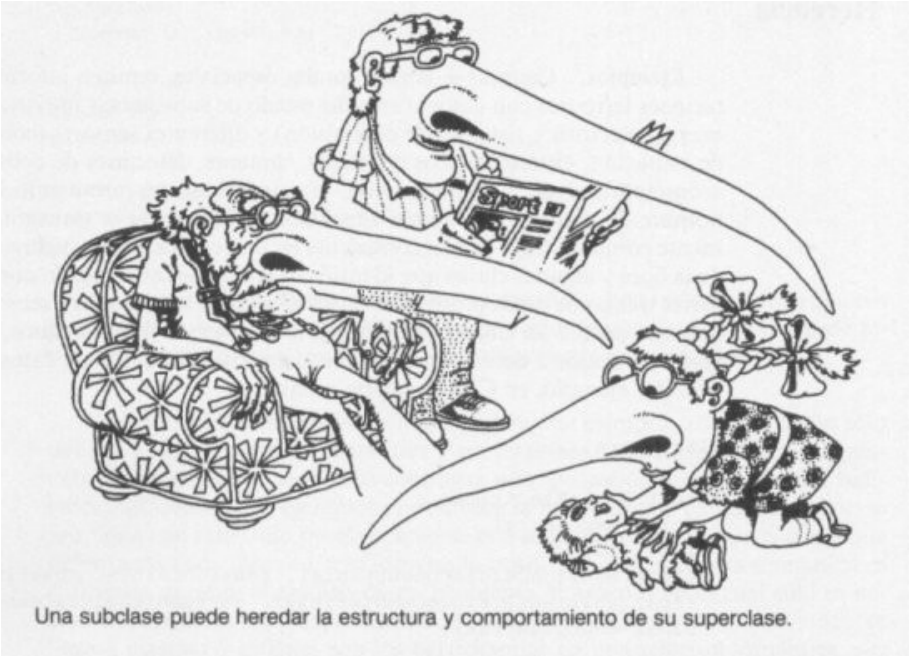
2.3 MODULARITAT

Aquesta propietat fa referència a la forma en la que els elements en programació orientada a objectes es troben organitzats en mòduls, facilitant així l'encapsulament i abstracció de la informació.

La modularitat permet dividir una aplicació en parts. Aquestes parts han de ser tan independents com puguen de la resta de mòduls així com també de l'aplicació principal. El motiu pel qual els mòduls han de ser el més independents possibles de l'aplicació és perquè així es podran reutilitzar en altres aplicacions més endavant.

2.4 JERARQUIA I HERÈNCIA

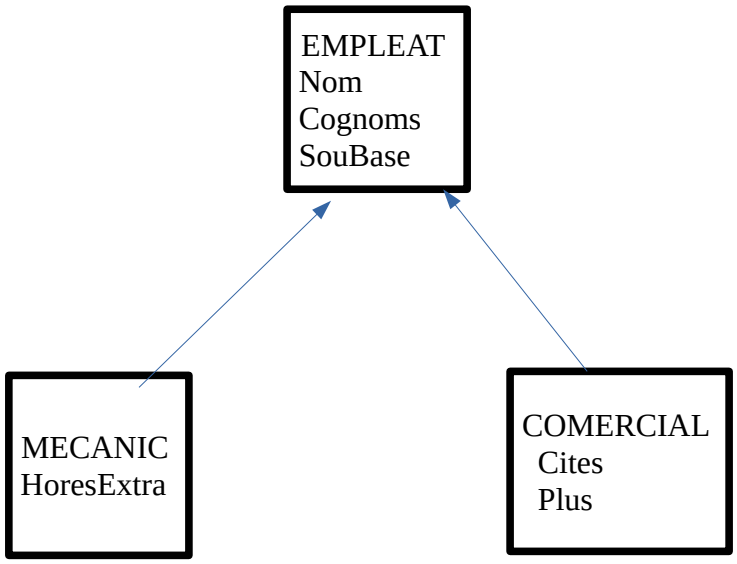
La jerarquia és la propietat de la programació orientada a objectes respecte de la relació o ordre existent entre les distintes classes i objectes. El mecanisme principal per tal d'ordenar les classes als nostres projectes és l'herència. Podem parlar d'herència entre classes quan elements d'una classe (superclasse) son heretats per una altra o inclòs els seus mètodes poden ser sobrecarregats.



Una subclase puede heredar la estructura y comportamiento de su superclase.

Suposem una aplicació d'un taller de vehicles en el qual treballen moltes persones però cadascuna fa una funció diferent al taller. Malgrat que cadascuna es dedique a un tasca concreta i no sàpiga res de la resta de feines que es fan al taller, tots els empleats són persones, és a dir, tots tenen un sèrie de característiques comuns: Nom, cognoms, DNI, data de naixement etc.. Mentre que al mateix temps hi ha algunes característiques especifiques per a cada tipus d'empleat

Imaginem que tenim dos classes d'empleats: els mecànics i els comercials. Dels primers hem de guardar informació del nombre d'hores extra que fa al mes mentre que dels segons hem de gestionar informació sobre les cites que té amb clients durant la setmana.



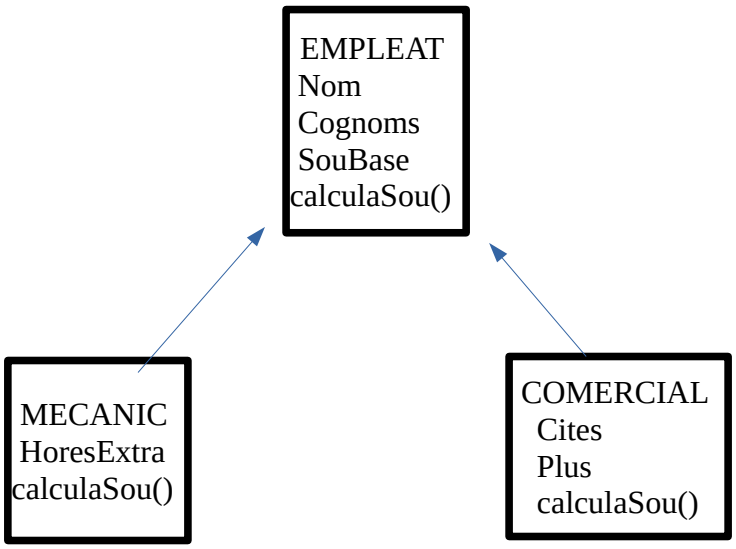
2.5 POLIMORFISME

Polimorfisme és una paraula grega que significa «moltes formes». Aquesta paraula és ideal per definir aquesta característica de la programació orientada a objectes ja que en POO podem tindre objectes amb el mateix nom que realitzen funcions distintes o implementen codi diferent.

El polimorfisme és un mecanisme que s'utilitza molt en POO juntament amb l'herència tal que quan una classe deriva d'una altra hereta les seues característiques i operacions. A la classe que hereta es pot redefinir qualsevol mètode heretat, de forma que en funció de la referència a objectes que usem a cada moment s'estarà executant un mètode o un altre.

Un mateix missatge enviat a objectes diferents donarà com a resposta eixides diferents, és a dir, si tenim diversos objectes amb mètodes similars, segons es referencie l'un o l'altre objecte el resultat final serà diferent.

Per exemple seguint amb l'esquema dels empleats de la nostra empresa suposem que la classe («superclasse») té un mètode que es diu «calcularSou», aquest mètode l'hereten tant el mecànic com el comercial però el seu càlcul seria diferent en un i l'altre. El sou del mecànic serà el sou base més el número d'hores extra multiplicades pel preu al que es paguen les hores extra, mentre que el sou del comercial seria el sou base més el plus.



2.6 REALITZACIÓ DE PROGRAMES ORIENTATS A OBJECTES

A partir d'ara, quan hajam de resoldre un problema hem de començar a pensar d'una forma diferent a la que estàvem fent fins ara. En programació orientada a objectes seguirem usant els mecanismes de programació estructurada com ara sentències repetitives, alternatives, etc.. tanmateix hem de plantejar un sistema on les dades presenten una major importància.

L'objectiu de l'orientació a objectes és plasmar la realitat de manera que programem els elements el més pròxim possible a com els veiem al seu entorn natural. Així, quan hajam d'encarar un problema d'aquest tipus:

- Ens preguntarem sobre aquells elements dels que realment es interessant emmagatzemar informació, és a dir, localitzarem els possibles objectes del nostre programa.
- De cada element hem de conèixer quines són les seues característiques o quines són les característiques que el representen.
- De cada element hem de conèixer les operacions a realitzar ja que cada conjunt de dades permetrà la realització d'un conjunt d'operacions.
- Finalment, desenvoluparem un entorn on utilitzarem els elements configurats.

Per exemple, a l'hora de desenvolupar l'aplicació per al un taller de reparacions de vehicles, podem resoldre les següents qüestions per determinar els elements que inclourem:

1. Quins elements del taller han de ser desats o s'han de tenir en compte a la nostra aplicació? Els vehicles i els empleats

2. De cada element, quines dades ens interessen? Dels vehicles: l'hora d'entrada al taller, danys que presenta, matrícula, marca i model, etc.. de cada empleat: nom i cognoms, adreça, sou, càrrec, reparacions associades, etc..

3. A cada element, quines operacions associarem? A un vehicle podem canviar-li l'oli, reparar la xapa, canviar el color, posar a punt, etc.. als empleats podem modificar-los el sou base, establir hores de treball etc..

4. Pel que fa a l'aplicació principal que utilitze aquests objectes, podem plantejar un programa que emmagatzeme de forma dinàmica els vehicles que van entrant al taller, així com també els empleats que treballen en ell estructurar la funcionalitat mitjançant un menú d'opcions que permeta utilitzar les operacions anteriorment plantejades per a cada objecte.

Activitat 1. Batalla naval

Fes un anàlisi semblant al de l'aplicació del taller però per al joc d'enfonsar la flota. Quants objectes diferents hi ha? Quines són les seues característiques principals? Quines accions poden realitzar?

3 CLASSES I OBJECTES

Una classe és la definició d'un objecte, és l'element que descriu els components d'un objecte de forma general. Diem que hem creat un objecte quan donem valors i utilitzem els components definits a la classe.

Per exemple, si continuem utilitzant el nostre taller de vehicles, una classe podria ser la classe cotxe, en la que es descriu de forma general aquest element de la vida quotidiana. S'establiria que un cotxe es caracteritza per:

- Tindre rodes
- Tindre matricula
- Tindre portes
- Se d'una marca i un model concrets.
- Etc..

Ara bé, si al nostre taller entra un Seat Arosa de tres portes, les característiques abans esmentades adopten certs valors:

- Quatre més la de recanvi.
- Matricula. 1234ACB
- 3 portes
- Seat. Model Arosa

Així tenim per un costat la classe cotxe i per l'altra l'objecte Seat Arosa.

Alhora de crear una classe definim el que denominem propietats o atributs i mètodes de classe.

- Atributs: una característica concreta d'un objecte de la vida quotidiana. Quan s'estableix un atribut a una classe, es defineix com quan creàvem variables als nostres programes estructurats, indicant tipus de dades i nom amb el que el reconeixem.
- Mètodes de la classe: donen funcionalitat a la classe, és a dir, reflecteixen les operacions que aquesta pot fer sobre els atributs. Els mètodes són similars a les funcions o procediments que creàvem en programació estructurada però orientats a modificar i actuar sobre les propietats d'una classe.

Un objecte dona valors concrets als atributs que defineix la classe i utilitza els seus mètodes. Quan un objecte utilitza un mètode concret es diu que aquesta enviant un missatge. D'alguna manera açò té lògica ja que estem donant una ordre a un component que reflecteix la realitat. Si a un taller hem de canviar el color d'un vehicle, direm canvia el color a l'operari que estiga treballant, li estem enviant un missatge de que ha de realitzar aquesta acció concreta.

A l'hora de enviar un missatge a un element concret d'un objecte utilitzarem l'operador punt, per exemple, arosa.canviaColor().

A sovint donem el nom d'estat als valors que en un objecte té en un moment concret de la vida del programa on s'està utilitzant. A més a més, quan creem un objecte, donem valors a les propietats de la classe, diem que estem instanciant aquesta classe.

4 DEFINICIÓ DE CLASSES EN JAVA

Una vegada ja sabem que són les classes i els objectes veurem com es defineixen classes i instancien objectes en el llenguatge Java. Definir una classe implica donar-li nom a ella i als elements que emmagatzemen la seua informació, així com també descriure els mètodes que realitzaran les accions considerades als objectes.

Les definicions o especificacions no constitueixen un codi de programa executable si no que s'utilitzen per assignar memòria on emmagatzemar els valors dels atributs que utilitza el programa i reconèixer els mètodes que aquesta utilitzarà; normalment es situen en arxius formant packages, utilitzant un arxiu per vàries classes relacionades.

```
class nomClasse {
   llista_de_membres
}
```

- **nomClasse:** definit per l'usuari i identifica la classe; pot incloure lletres, números i subratllats com qualsevol identificador.
- **llista_de_membres:** són els mètodes i atributs de la classe, etc..

Més concretament, una classe es podria definir de la següent manera:

```
class nomClasse {
   llista_de_constants;
   llista_de_atributs;
   constructor_de_classe;
   constructor_de_còpia;
   mètodes;
}
```

En realitat si la definició de les parts de la classe es fera en un altre ordre, no afectaria a la compilació i execució d'aquesta però sí a la lectura del codi i el faria més complicat d'interpretar.

Exemple de classe: per tal d'il·lustrar com es defineix una classe amb un exemple, definirem la classe 'punt'. Aquesta classe determina les coordenades a les quals es troba un punt concret: x i y;

```
class Punt {
    private int x;
    private int y;
    public Punt () {
        x = 0;
        y = 0;
    }
    public Punt (int a, int b) {
        x = a;
        y = b;
    }
    public int LlegirX () {
        return x;
    }
    public int LlegirY () {
        return y;
    }
    public void FixarX(int valor) {
        x = valor;
    }
    public void FixarY (int valor) {
        y = valor;
    }
}
```

Una vegada tenim la definició de la classe punt, instanciarem un punt concret que es troba a les coordenades 10 i 5

```
Punt a = new Punt(10, 5);
```

4.1 VISIBILITAT DELS MEMBRES DE LA CLASSE

Un principi fonamental en la programació orientada a objectes és l'ocultació de la informació, açò significa que no es pot accedir per mètodes externs de la classe a determinada informació interna. El mecanisme principal per aconseguir-ho és posar-los dins d'una classe i fer-los privats i així només es podrà accedir a aquests des de dins de la classe

Existeixen tres diferents especificacions d'accés o modificadors de visibilitat:

- **public (+):** un mètode o atribut té una visibilitat pública quan totes les demès classes poden accedir a ells, bé siga una altra classe o una subclasse.

- **Private (-):** sols es pot accedir a ells des del propi codi de la classe.
- **Protected (#):** només des del propi codi de la classe o de les seues subclasses es pot accedir.

4.2 ATRIBUTS

Els atributs són les característiques individuals que diferencien un objecte d'un altre i determinen la seua aparença, estat o altres qualitats. Els atributs es desen en variables anomenades variables d'instància i cada objecte particular pot tindre valors distints per a aquestes variables. Per exemple, si hem de definir una persona en base a les seues característiques individuals que diferencien a una persona d'una altra en el món real, què escolliríem? Segurament: estatura, color de la pell, color dels ulls, color del pel, etc.. però segurament també altres característiques no tan visualment evidents com podrien ser: nom, cognoms, data de naixement, etc.. i altres més «administratives» com podria ser el DNI. Així ens quedaria la classe persona:

```
public class Persona {
    private String DNI;
    private String nom;
    private String cognoms;
    private String dataNaixement;
    private double estatura;
    private double pes;
    private String colorUlls;
    private String colorPel;
}
```

Les variables d'instància també anomenades membres són declarades a la classe però els seus valors son canviats i fixats a l'objectes ja que una classe no és més que un esquema que defineix com seran les persones al nostre programa.

A més de les variables d'instància hi ha variables de classe, les quals s'apliquen a la classe i a totes les instàncies. Per exemple, el número de rodes d'un cotxe és el mateix per a tots els objectes cotxe.

4.3 MÈTODES

Els mètode són aquelles accions que pot realitzar la classe que estem definint. Bàsicament el que fa un mètode és modificar l'estat de l'objecte mitjançant el canvi del valor dels seus atributs. Continuant amb la classe persona que hem definit al punt anterior, caldria que ens férem les següents qüestions: quines accions són les que pot realitzar la nostra persona? En que afecta eixes accions als atributs que hem escollit per definir-la?

A la qüestió primera diríem que una persona pot: caminar, botar, córrer, menjar, créixer, tintar-se els cabells, etc.. Pel que fa a la segon qüestió, podríem dir que només modifiquen atributs: menjar, augmentant de pes; créixer, modificant l'estatura i tintar-se els cabells, canviant el color del pel. Així la classe persona podria quedar de la següent forma:

```
public class Persona {
    // Atributs
    private String DNI;
    private String nom;
    private String cognoms;
    private String dataNaixement;
    private double estatura;
    private double pes;
    private String colorUlls;
    private String colorPel;

    // Mètodes
    public void creixer ( ) { estatura ++; }
    public void menjar ( ) { pes ++; }
    public void tintarse (String colorNou) { colorPel = colorNou; }
}
```

Fixeu-se que mentre els atributs de la classe tenen un modificador de visibilitat private, els mètodes el solen tindre public. Encara que no necessàriament ha de ser així sempre però és molt habitual.

Els mètodes a Java sempre són membres d'una classe, no hi ha mètodes o funcions fora d'aquestes. La seua implementació s'inclou dins del cos de la classe.

Exemple: A continuació modelarem una sèrie de classes geomètriques sobre les quals farem una sèrie de càlculs. Utilitzant un llenguatge de programació orientada a objectes com és Java, modelarem un triangle i un quadrat.

Primer la classe triangle.

```
public class triangle {
    // Membres de la classe
    int iTotalCostats;
    private double dArea;
    private double dCostat;
    private double dAltura;
    // Constructor de la classe
    public triangle () {
        iTotalCostats = 3;
        dCostat = 1;
        dAltura = Math.sqrt(3*dCostat)/2;
    }
    public triangle (double dLong) {
        iTotalCostats = 3;
        dCostat = dLong;
    }
    public double getArea ( ) {
        return (dCostat * dAltura) / 2;
    }
}
```

A continuació la classe quadrat

```
public class quadrat {
    // Membres de la classe
    int iTotalCostats;
    private double dArea;
    private double dCostat;
    // Constructor de la classe
    public quadrat () {
        iTotalCostats = 4;
        dCostat = 1;
    }
    public quadrat (double dLong) {
        iTotalCostats = 4;
        dCostat = dLong;
    }

    public double getArea ( ) {
        return dCostat * dCostat;
    }
}
```

4.4 INSTÀNCIES

La declaració d'una classe no és més que la definició d'una realitat utilitzant un llenguatge de programació orientada a objectes. El següent pas seria fer ús d'aquesta definició o classe, és a dir, instanciar o crear un objecte a partir d'aquesta classe.

Per tal d'instanciar una classe s'utilitza la següent sentència:

```
nom_classe nom_instancia= new nom_classe(paràmetres);
```

La paraula «new» s'encarrega de reservar espai en memòria i torna una referència a l'adreça de memòria on s'ha emmagatzemat l'objecte.

- **nom_classe:** en el primer cas és el nom de la classe que es vol instanciar. A la segon part es refereix al constructor de la classe que ha de ser igual que el nom d'aquesta. El constructor és un mètode especial de la classe i com a mètode que és pot rebre paràmetres.
- **nom_instancia:** identificador amb el que s'identificarà l'objecte instanciat.

L'espai reservat a memòria per un objecte dependrà de la quantitat d'elements o membres que continga. Quan un objecte no està siguen referenciat es treu de memòria per tal d'alliberar espai pel «garbage collector»

Quan creem un objecte d'una classe, cada atribut serà inicialitzat amb els valors per defecte del tipus de dades utilitzat o bé pel valor establert al constructor.

4.5 CONSTRUCTORS

Un constructor és un mètode membre d'una classe que s'executa de forma automàtica quan es crea un instancia d'aquesta. El constructor d'una classe es diferencia de la resta perquè s'ha d'anomenar igual que la classe (respectant majúscules i minúscules) i mai s'especifica a la seua definició un tipus de dades a retornar encara que siga void.

El constructor admet paràmetres, sent un mètode que es pot sobrecarregar, és a dir, podem tindre diversos constructors amb un nombre diferent de paràmetres o diferents tipus de dades. Els constructors tenen com a finalitat principal la inicialització de les variables o atributs de la classe.

No és obligatori definir un constructor a cada classe, encara que es sol fer. Així establim els valors que desitgem per quan un atribut de forma automàtica a la creació de l'objecte, en lloc de fer-ho en un altre moment mitjançant altres mètodes.

```
public class empleat {
    // Atributs de la classe
    private String strNom;
    private String strCognom1;
    private String strDNI;
    public double dSouBase;

    // Constructors de la classe
    public empleat ( ) {
        strDNI = «00000000Z»;
    }

    public empleat ( String strNIF ) {
        strDNI = strNIF;
    }

    public empleat ( String NIF, String nom, String C1 ) {
        strDNI = NIF;
        strNom = nom;
        strCognom1 = C1;
    }
}
```

4.6 ACCÉS ALS MEMBRES DE LA CLASSE

Podrem accedir als membres d'una classe en funció de la seua visibilitat i el lloc on es desitge accedir. Els modificadors de visibilitat són: public, private, protected i internal.

Per accedir a un membre de la classe utilitzem el «.» (punt): NomClasse.membre

Basant-nos en la classe Persona dels punts anteriors farem un programa que instàncie un objecte Persona i veurem com s'accedeix als membres de la classes

Suposem que persona té un constructor de la classe al qual li passem els següents paràmetres: DNI, nom, cognom, estatura, pes i color del pèl.

```
public class programa {
    public static void main (String [] agrs) {
        Persona vicent;
        vicent= new Persona("11111111A", "Vicent", "Garcia", "11/22/33", 180, 80, "blau", "roig");
        vicent.menjar ();
        vicent.creixer ();
        vicent.tintarse ("morat");
    }
}
```

Tanmateix no podríem accedir als membres privats de la classe des de fora, per exemple, no podríem fer vicent.nom= «Antonio» ja que nom és un membre privat.

5 LA PARAULA RESERVADA THIS

La paraula reservada «this» s'utilitza per fer referència a l'objecte actual. Si estàs dissenyant una classe, «this» fa referència a la pròpia classe.

Quan es crida a un mètode, es passa automàticament un argument implícit que és una referència a l'objecte invocat, és a dir, l'objecte sobre el que es crida al mètode. Aquesta referència és «this». Amb els atributs de la classe també passa el mateix.

Per comprendre-ho millor, mirem el següent exemple:

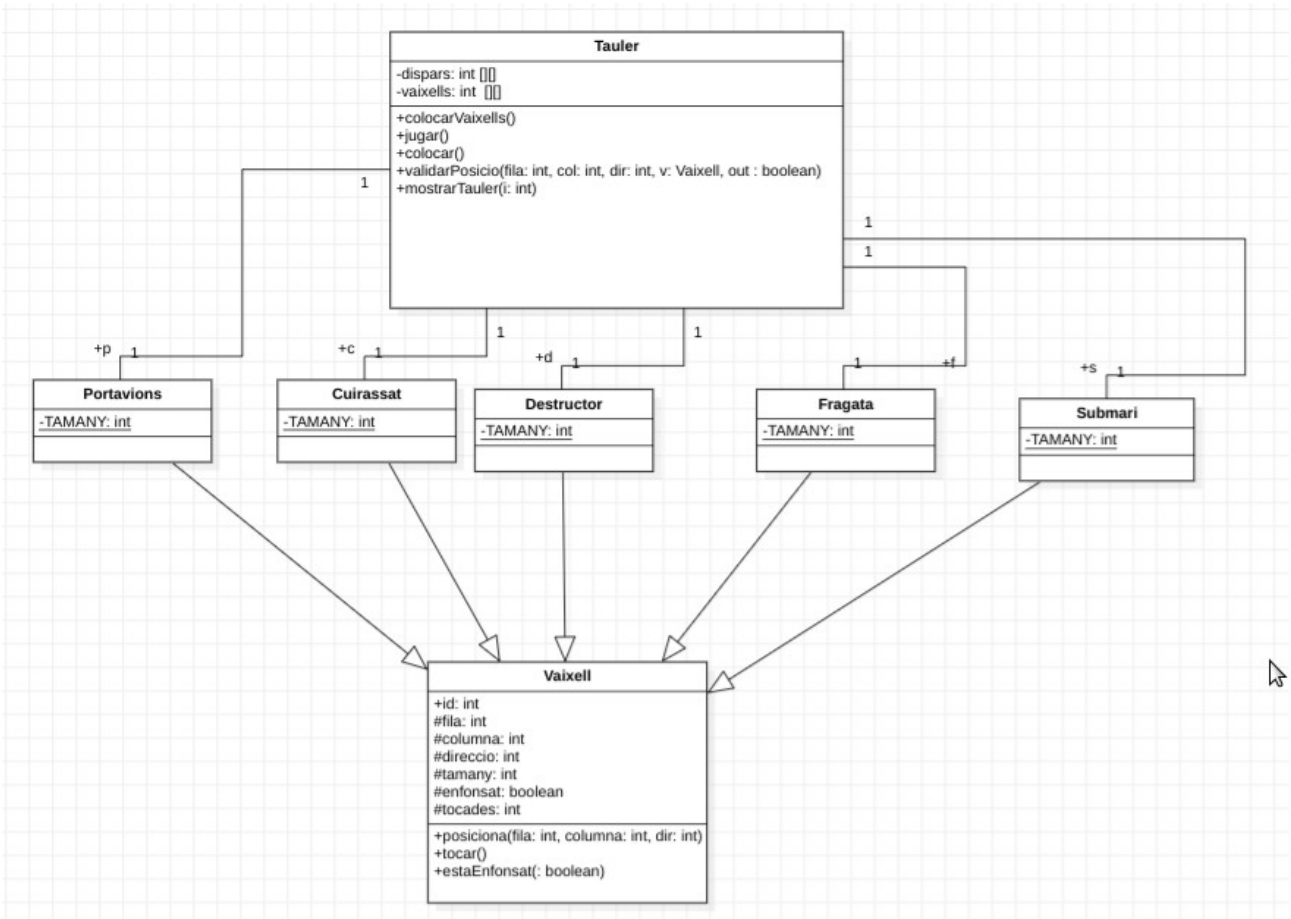
```
public class Persona {
    public String nom;
    public String cognom;
    public String DNI;

    public Persona (String nom, String cognom, String DNI ) {
        this.nom = nom;
        this.cognom = cognom;
        this.DNI = DNI;
    }

}
```

Aquesta paraula clau sol utilitzar-se quan hi han bé mètodes o atributs (membres) a dues classes diferents amb el mateix nom per exemple.

6 DESENVOLUPAMENT D'UN PROJECTE EN JAVA

Vista ja un poc la teoria de com és el paradigma de programació orientada a objectes i continuant amb l'activitat d'anàlisi del joc de «Batalla Naval», ha arribat el moment de desenvolupar un projecte utilitzant la metodologia POO amb el llenguatge de programació Java.

Donarem per suposat que les fases d'anàlisi i disseny ja han sigut realitzades i disposem de l'esquema UML amb el diagrama de classes del nostre joc.



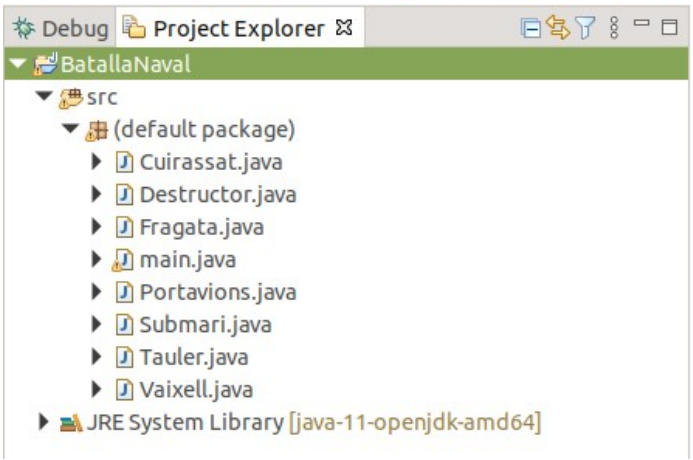
6.1 PER ON COMENCEM

Si disposem d'una eina de modelat UML, podríem a partir del diagrama de classes que ens han passat generar el codi en Java que ens serviria per poder començar a programar ja el seu comportament.

Existeixen múltiples eines lliures que s'encarreguen de fer aquesta tasca automàticament com són: StarUML, ArgoUML, Umbrello i altres extensions que podem afegir a alguns IDEs com són Eclipse. Aquestes eines el que fan és

generar els esquelets de les classes amb tots els membres especificats al diagrama, és a dir: constructors, atributs i mètodes. La qual cosa ens estalviarà un munt de feina.

Una vegada ja tenim cadascun dels esquelets de les nostres classes en fitxers, hauríem de crear un nou projecte i afegir-li aquestes classes. Aquesta part es podria fer amb un IDE tipus Eclipse, NetBeans o Visual Studio Code per exemple.



Arribats a aquest punt ja tenim preparat el nostre projecte i ja podem començar a «picar codi»

6.2 LA FUNCIÓ MAIN A JAVA

El mètode main a Java és un estàndard utilitzat per la maquina virtual de Java o JVM per iniciar la execució de qualsevol programa escrit en Java. Aquest mètode es coneix com el punt d'entrada de l'aplicació Java.

En altres paraules, per poder executar la nostra aplicació «Batalla Naval» haurem de tindre un mètode main que instancie i utilitze les classes que hem creat a partir del diagrama de classes que ens han passat. Aquest mètode main ha d'estar dins d'una classe, per tant, ens creem una classe nova que es diga «Joc» i en ella podrem implementar el mètode main.

```
4  public class Joc {
5      public static void main (String [] args) {
6
7
8      }
9
10 }
11
```

Main ha de ser «public» perquè és el punt d'entrada a la nostra aplicació i per tant ha de ser accessible de fora de la classe i ha de ser «static» perquè és un

mètode que s’ha de poder cridar sense necessitat d’instanciar un objecte de la classe Joc.

Activitat 2. Tenda d'electrodomèstics

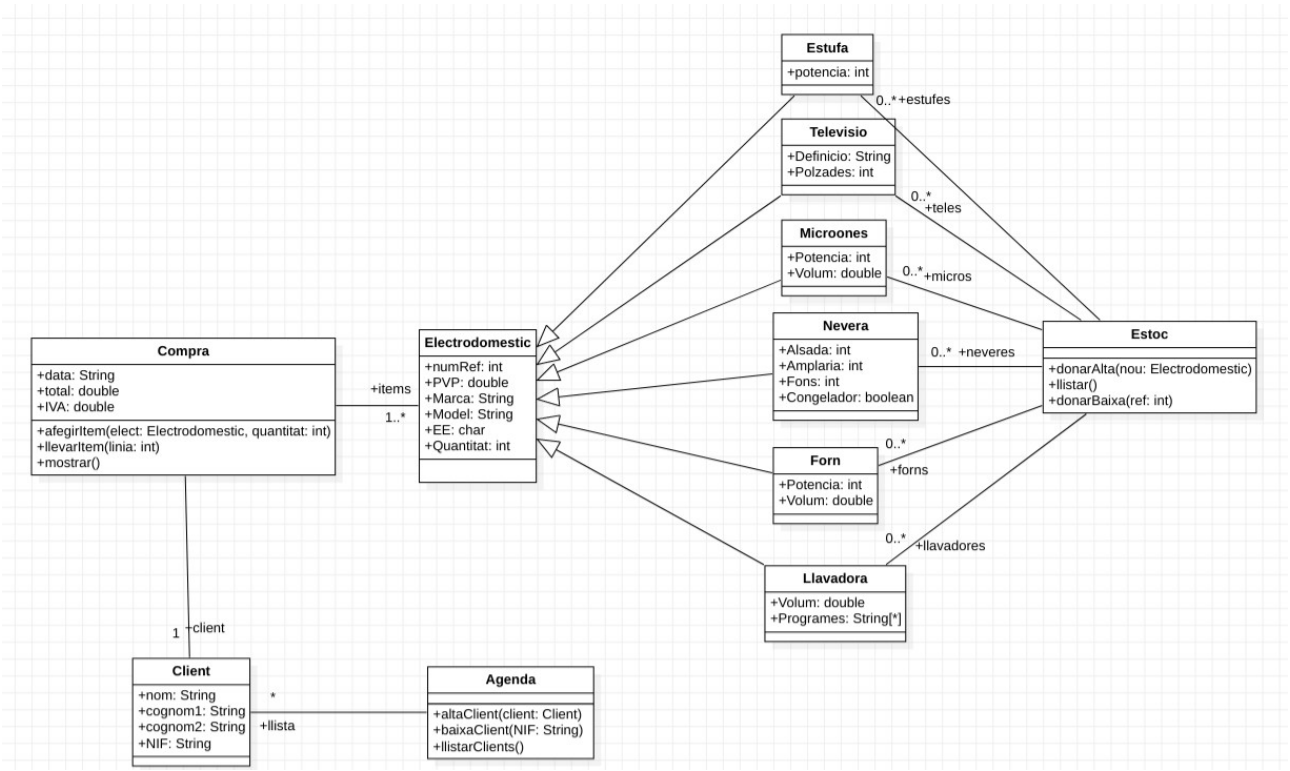
Manolo és el propietari d'una tenda d'electrodomèstics en la té: televisors, neveres, microones, forns, llavadores, estufes i altres. A Manolo li agradaria tindre una aplicació en la que puga emmagatzemar la informació relativa a tots els electrodomèstics que té a la tenda en estoc així com també li agradaria tindre la informació guardada de tots els clients en una agenda i quines són les compres que li han realitzat.

De cada electrodomèstic vol saber coses com la marca, el model, el número de referència, eficiència energètica i ja més específicament la potència dels microones, el volum i l'altura de les neveres així com si tenen o no congelador, les polzades i tipus de definició dels televisors, potència i volum del forn, programes de les llavadores i capacitat, potencia de les estufes etc...

De cada client vol saber el seu nom, cognoms, adreça, DNI i tots els electrodomèstics que ha comprat a la tenda. També seria interessant poder guardar un inventari de la disponibilitat d'electrodomèstics que hi han actualment.

Les accions que podrà realitzar la nostra aplicació seran: donar de alta i baixa electrodomèstics i clients, comprar electrodomèstics per part dels clients, consultar l'estoc etc..

El diagrama de classes resultant de l'anàlisi anterior podria ser el següent:



Activitat 3. Sèries i pel·lícules

Imagina que volem fer una aplicació sobre sèries i pel·lícules de les distintes plataformes d'streaming que existeixen en l'actualitat. Aquesta aplicació tindrà usuaris que es connectaran mitjançant unes credencials (nick i contrasenya) i cada usuari podrà afegir sèries o pel·lícules, a banda de posar qualificacions amb comentaris als continguts. El nick de cada usuari d'aquesta aplicació serà el seu correu electrònic, d'aquesta forma ens assegurem un nom únic per cada usuari. També s'enregistrarà per cada usuari el seu nom i cognoms.

De cada pel·lícula ens agradaria saber: títol, any, país de producció, genere i resum de la mateixa. Els generes disponibles seran: acció, aventures, animació, comèdia, documental, drama, terror, musical, romàntica, ciència ficció.

A banda també s'haurà de guardar informació relativa a les actrius i actors que intervenen així com també el director. De les quals haurem de saber el seu nom i cognoms així com també la seua nacionalitat i l'any de naixement.

Pel que fa les sèries, a banda de la informació que també tenen les pel·lícules, també ens agradaria saber el número de temporades que té, de quin any és la temporada i quants capítols té amb el seu títol i breu sinopsis.

Finalment cada usuari del nostre sistema podrà afegir tant sèries com pel·lícules i podrà qualificar-les (de 0 a 10) a banda d'acompanyar la qualificació d'un comentari al respecte.

7 MÈTODES, VARIABLES I BLOCS STATIC

La paraula reservada «static» en Java és un modificador que en Java es pot aplicar a variables, mètodes, classes i blocs de codi

7.1 VARIABLES ESTÀTIQUES

Una variable estàtica (static) és una variable que pertany a la classe en la que va ser declarada i s'inicialitza només una vegada a l'inici de l'execució del programa, la característica principal d'aquest tipus de variables és que es pot accedir directamen amb el nom de la classe sense necessitat d'instanciar un objecte. A banda també:

- És una variable que pertany a la classe (variable de classe) i no a l'objecte.
- Les variables static s'inicialitzen només una vegada, a l'inici de l'execució. Aquetses variables s'inicialitzaran primer abans de la inicialització de qualsevol variable d'instància.

7.2 MÈTODES ESTÀTICS

Un mètode estàtic només pot accedir a les variables o tipus de dades estàtiques i no pot accedir a les dades no estàtiques.

- Un mètode estàtic només pot accedir a dades estàtiques. No pot accedir a dades no estàtiques (variables d'instància)
- Un mètode estàtic pot cridar només a mètodes estàtics i no pot invocar un mètode no estàtic a partir d'ell.
- Un mètode estàtic es pot accedir directament pel nom de la classe i no es necessita crear un objecte per accedir al mètode.
- Un mètode estàtic no pot fer referència a «this» o «super»

7.3 BLOCS ESTÀTICS

El bloc estàtic és un bloc d'instruccions dins de la classe (static { ... }) que s'executa quan una classe es carrega per primera vegada en la màquina virtual de Java (JVM). Bàsicament un bloc estàtic inicialitza variables de tipus estàtic dins d'una classe, de la mateixa forma que un constructor de classe ajuden a inicialitzar les variables d'instància, un bloc estàtic inicialitza les variables tipus static de la classe.

8 SOBRECÀRREGA D'OPERADORS

En Java és possible sobrecarregar mètodes, és dir, definir dos o més dins de la mateixa classe, que comparteix nombre i que les declaracions dels seus paràmetres són diferents; la sobrecàrrega és una forma de polimorfisme.

En les cridades als mètodes sobrecarregats, el compilador determina quin és el mètode invocat basant-se en el nombre i tipus d'arguments passats; per tant, els mètodes sobrecarregats han de diferir en nombre i tipus de paràmetres. Quan Java troba una trucada a un mètode sobrecarregat, s'executa la versió del que té paràmetres (número i tipus) que coincideixen amb els arguments utilitzats en la trucada.

8.1 Sobrecarrega de constructors

En Java es possible sobrecarregar mètodes, és dir, definir dos o més dins de la mateixa classe, que comparteixen nombre i que les declaracions dels seus paràmetres són diferents; la sobrecàrrega és una forma de polimorfisme.

La classe descriu un conjunt d'objectes amb les mateixes propietats i comportament; quan l'objecte es crea, es inicialitza amb valors predeterminats o amb els que es transmeten en el moment de la instància; el mètode que realitza la inicialització de l'objecte és el constructor, aquest té el mateix nombre que la classe i no té tipus de retorn.

A més de la sobrecàrrega de mètodes normals, es poden sobrecarregar els constructors; estos últims normalment es sobrecarregaran a la majoria de les classes creades, encara que no es regla; fins i tot una classe pot definir-se sense constructor i, per ende, sense arguments.