

# Arquitectura en 3 capes i Orientació a Objectes:

Alicia Ageno, Ricard Gavaldà i Josep Ibarz

Creació: març 2002

Darrera revisió: 24 octubre 2005

L'exemple és d'Allen Peralta, 13 gener 1997

## 1. L'arquitectura en tres capes

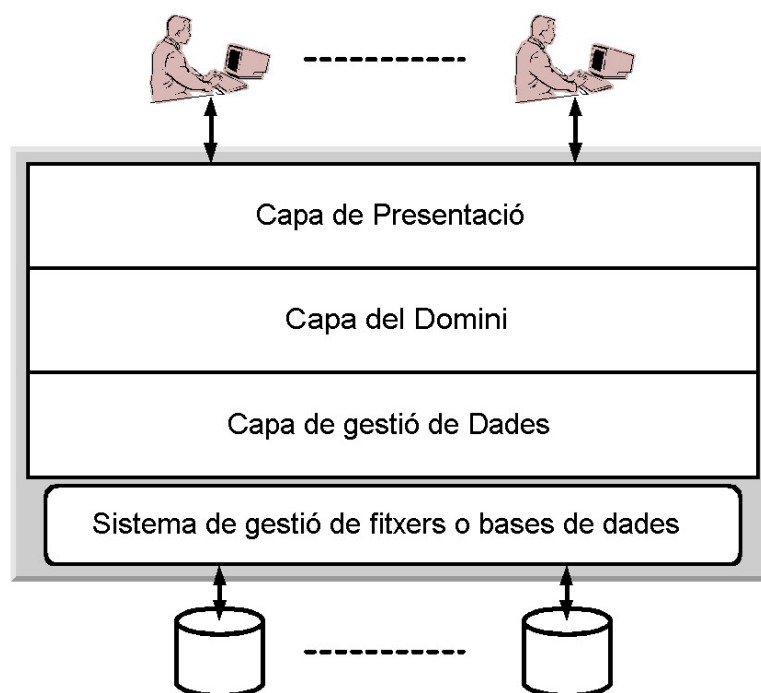
L'arquitectura del *software* és una descripció dels subsistemes i components computacionals d'un sistema *software*, i de les relacions entre ells. És el resultat de l'activitat de disseny arquitectònic del *software*. En aquesta activitat se solen usar patrons de disseny cada un d'ells adequats per un tipus de problemes i amb unes característiques específiques.

El patró de disseny (o estil) en capes té com a gran avantatge que facilita la canviabilitat (que vol dir: extensibilitat, portabilitat, mantenibilitat i reestructurabilitat) i la prova. Com a inconvenient, fa difícil d'obtenir l'eficiència òptima en afegir feina innecessària o redundant.

Les crides (i retorn) en un patró en capes han de complir:

- Els components (objectes i classes) s'agrupen en capes.
- La comunicació (relacions entre classes, missatges entre objectes) solament es produeix entre elements de la mateixa capa o de capes contigües.

L'arquitectura en 3 capes (en anglès, *Three-Tier Architecture*) segueix aquest esquema:



La capa de presentació sap com presentar les dades a l'usuari, però ignora quines transformacions cal fer per donar resposta a les peticions de l'usuari.

1. Es relaciona amb els usuaris: rebent-ne esdeveniments i presentant-los respostes i resultats.
2. Es relaciona amb la capa de domini: passant-li els esdeveniments externs (crides a accions) i consultes, i rebent-ne les respostes i resultats.
3. S'ocupa de:
  - Assabentar-se de les peticions dels usuaris.
  - Ordenar l'execució d'accions.
  - Comunicar els resultats de les accions als usuaris.
  - Tractar finestres, botons, diàlegs, menús i llistats.

La capa de domini sap com satisfer les peticions de l'usuari, però ignora on es guarden les dades i com es presenten a l'usuari.

1. Es relaciona amb la capa de presentació: passant-li les respostes i resultats, i rebent-ne els esdeveniments externs (crides a accions) i consultes.
2. Es relaciona amb la capa de gestió de dades: passant-li les operacions de consulta i modificacions de dades, i rebent-ne les respostes i resultats.
3. S'ocupa de:
  - Assabentar-se dels esdeveniments.
  - Controlar-ne la validesa.
  - Canviar l'estat del domini.
  - Executar les accions encomanades.
  - Assabentar-se de les consultes.
  - Obtenir-ne el resultat.
  - Comunicar la resposta.

La capa de gestió de dades sap on i com estan emmagatzemades les dades, però ignora com tractar-les.

1. Es relaciona amb la capa de domini: passant-li les respostes i resultats, i rebent-ne les operacions de consulta i modificació de dades.
2. Es relaciona amb el sistema de gestió de bases de dades o de fitxers: passant-li les operacions de consulta i modificacions de dades en el format i llenguatge adequats, i rebent-ne les respostes i resultats.
3. S'ocupa de:
  - Permetre-li al domini d'ignorar on són les dades.
  - Permetre que determinats objectes del domini siguin persistents.

Amb aquesta estructura s'aconsegueix:

- Que un canvi en la representació persistent de les dades (per exemple, un canvi en el sistema gestor de bases de dades o fitxers), normalment, només afecti a la capa de gestió de dades (diem normalment perquè l'estructura adequada per alguna solució amb determinats gestors de bases de dades pot afectar la capa de domini).
- Que un canvi en la interfície del programa (per exemple, un canvi en el sistema de finestres o en els perifèrics usats per a la comunicació amb l'usuari) afecti només la capa de presentació.
- Que la capa de domini, que encapsula la majoria de la lògica del programa, sigui força independent dels canvis de plataforma, sistema operatiu, etc.

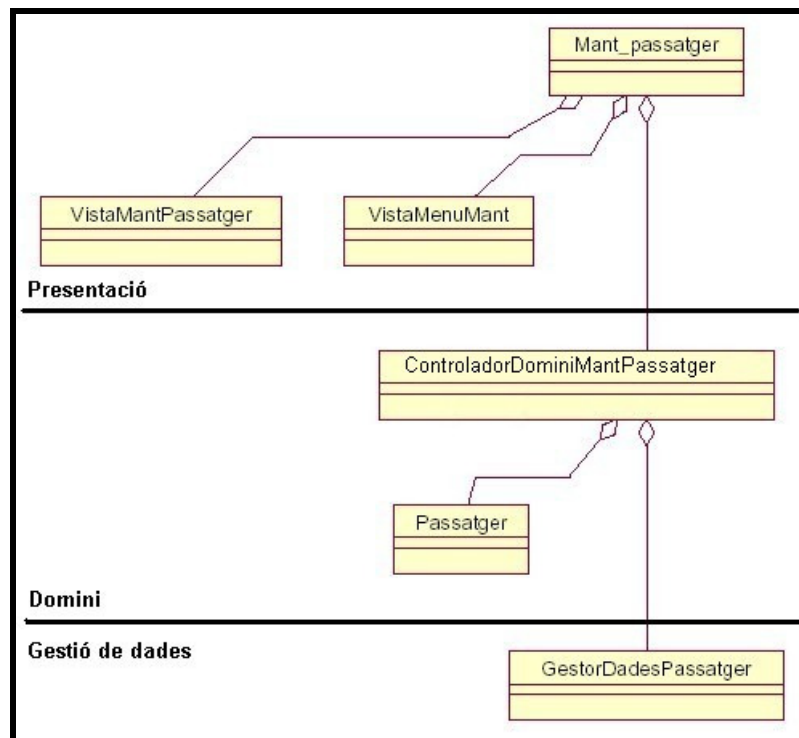
## **2. Estructura d'un programa en tres capes: conceptes addicionals**

- En moltes ocasions ens convindrà disposar d'unes classes que tinguin la funció d'aglutinar-ne d'altres, d'una banda per coordinar la funcionalitat d'aquestes classes sobretot la conjunta i d'una altra per facilitar descarregar-les de funcionalitats. Aquestes classes s'anomenen *controladors*.

Com que ens convé que a les classes del model (o domini del programa) només hi hagi les operacions més bàsiques (funcionalitats de consulta i actualització i d'altres que siguin molt típiques, com el descompte en les vendes) a més de les dades (atributs), perquè això en facilita la reutilització. Es tracta de posar els aspectes funcionals més concrets de l'aplicació a banda, és a dir, en controladors. En conseqüència, moltes vegades la lògica de la/es funcionalitat/s (*use case*) estarà en els controladors.

Pensant altre cop en la reutilització, en aquest cas, de les vistes, també resulta més pràctic separar l'aspecte de la vista (disseny de la pantalla o llistat) dels mètodes que les gestionen i que controlen el comportament de la interfície. Convindrà, doncs, disposar-los en controladors.

Per tant, hi pot haver controladors tant a la capa de presentació com a la de domini. El propi programa, l'objecte programa, sol ser un controlador de la capa de presentació que ofereix a l'usuari la funcionalitat del programa. Així, l'estructura seria:



- Recordeu que només s'implementen els *links* (a les relacions d'associació) en el sentit que interessa navegar.
- Per poder garantir que sempre es fa una validació homogènia dels valors dels atributs, convé centralitzar-la, és a dir, que l'actualització es faci amb un únic mètode especialitzat. Això, a més, facilita el manteniment d'aquesta validació. Per això convé ocultar els atributs, fent-los sempre privats (només visibles des del propi objecte) i generar per a cada un d'ells els dos mètodes següents:

```

➤ get_atribut_x // retorna el valor de l'atribut x.
➤ set_atribut_x // valida i, si s'escau, actualitza el valor d'x.
  
```

- Els filtres dels valors dels elements del model (coherència de les dades) és millor de posar-los a l'estructura del model. Així que convé que el mètode `set_atribut_x` comprovi la validesa del valor subministrat i que retorni informació sobre aquesta validesa (en aquest cas retorna `cert` quan el valor de `nou_valor` és vàlid):

```

public boolean set_atribut_x (tipus_de_x nou_valor)
{
    if (<filtre>)
    {
        atribut_x = nou_valor; // Atenció amb tipus_de_X
        return true;
    }
    else
        return false;
    // if
} // set_atribut_x ()
  
```

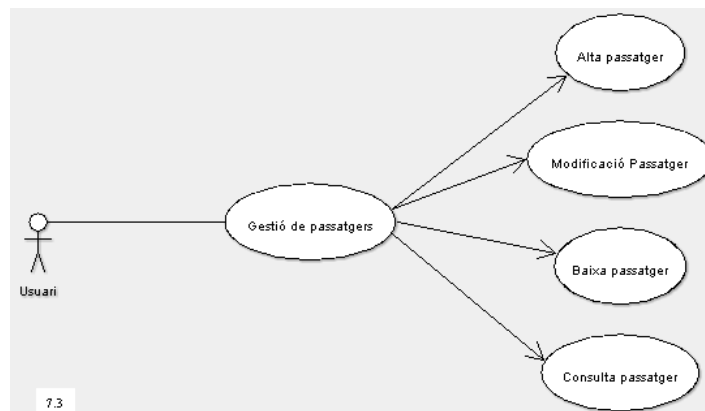
Qualsevol classe que faci ús d'aquest mètode usa el filtre (que ha de ser una funció booleana que retorni `cert` quan el valor del `nou_valor` es pugui acceptar com a valor de l'atribut `x`), així que no caldrà comprovar més el nou valor de l'atribut.

- El gestor de persistència o de disc sol tenir una estructura de dades paral·lela a la del model, convenientment adaptada per emmagatzemar correctament els *links* que implementen les associacions i les composicions. Guardar els *links* d'un objecte pot ser responsabilitat de la mateixa operació que guarda l'objecte, o bé poden haver-hi operacions específiques per guardar els *links* i és responsabilitat del programador utilitzar-les per mantenir el model actualitzat.

### 3. Exemple d'estructuració en tres capes: un ABM

Prenem com a exemple d'arquitectura en tres capes un programa que ha de fer el manteniment (altes, baixes i modificacions, és a dir, l'ABM) dels passatgers d'un vol. Podria ser part del programa de gestió d'una agència de viatges.

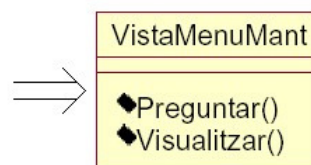
Malgrat que el més probable és que el vostre programa sigui regit per esdeveniments, estimem que resulta més clar un exemple on aplicar l'arquitectura de tres capes d'un programa amb un comportament seqüencial, on és el programa qui du la iniciativa del diàleg i el control del flux d'execució. Per aquesta raó el programa que usem com a exemple és d'aquest tipus. Partirem del següent cas d'ús:



Suposem que s'ha dissenyat pantalles per triar l'opció de manteniment (cas d'ús *Gestió de Passatgers*) i per a la lectura de dades d'un passatger (casos d'ús *Alta Passatger* i *Modificació Passatger*). Suposem també que el seu control s'ha delegat a dues classes *VistaMenuMant* i *VistaMantPassatger* respectivament.

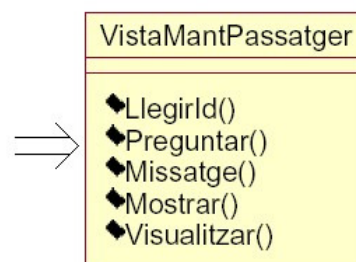
Menú:

<b>Manteniment del Passatger</b> 1. Alta 2. Baixa 3. Modificació 4. Consulta
Pregunta:
Missatges



Formulari de dades:

<b>Dades del passatger</b> Codi: _____ 1: Nom: _____ 2: Adreça: _____ :
Pregunta:
Missatges



En Java es podria implementar la capa de presentació del manteniment de passatgers amb una classe controlador `mant_passatger` com aquesta:

```

public class mant_passatger // controlador de la capa de presentació
{
    VistaMenuMant VMM;
    VistaMantPassatger VMP;
    ControladorDominiMantPassatger CDMPS;

    public mant_passatger ()
    {
        char OPC;
        VMM = new VistaMenuMant ();
        VMM.visualitzar ();
        OPC = VMM.preguntar ("Quina opció vols?", '1', '2', '3', '4', '0');
        while (OPC != '0')
        {
            switch (OPC)
            {
                case '1': alta ();
                    break;
                case '2': baixa ();
                    break;
                case '3': modifica ();
                    break;
                case '4': consulta ();
                    break;
            } // switch
            VMM.visualitzar ();
            OPC = VMM.preguntar ("Quina opció vols?", '1', '2', '3', '4', '0');
        } // while
    } // mant_passatger ()

    private void alta ()
    {
        char OPC;
        id_pass ID;
        passatger PS;
        VMP = new VistaMantPassatger ();
        CDMPS = new ControladorDominiMantPassatger ();
        VMP.visualitzar ();
        ID = VMP.LlegirId (); // l'obté i el crea (pot validar la sintaxi d'ID)
        while (ID != null)
        {
            if (CDMPS.existent(ID)) // CDMPS usará el gestor de dades
                VMP.missatge ("Passatger existent");
            else
            {
                PS = new passatger ();
                if (PS.set_ID (ID))
                {
                    obtenirLaRestaDeDadesDelPassatger (PS); // obté i valida PS
                    VMP.mostrar (PS);
                    OPC = VMP.preguntar ("Confirma (S/N): ", 'S', 'N');
                    if (OPC = 'S')
                        CDMPS.afegir (PS); // Comunica la validació de l'usua-
                                        // ri, ja es pot afegir, CDMPS usa
                                        // el gestor de dades si convé.
                }
                else
                    VMP.missatge ("codi de passatger no vàlid");
            } // if
        } // if
        VMP.visualitzar ();
        ID = VMP.LlegirId (); // l'obté i el crea (pot validar la sintaxi d'ID)
    } // while
    } // alta ()
}

```

```

private void obtenirLaRestaDeDadesDelPassatger (passatger PAS) // versió molt
{ // rudimentària
    String nom, adreça, ...;
    ...;
    boolean valid;
    while (!valid)
    {
        nom = VMP.read_nom ();
        valid = PAS.set_nom (nom);
        adreça = VMP.read_adreça ();
        valid = PAS.set_adreça (adreça) && valid;
        ...
    } // while
} // obtenirLaRestaDeDadesDelPassatger ()

...
} // mant_passatger

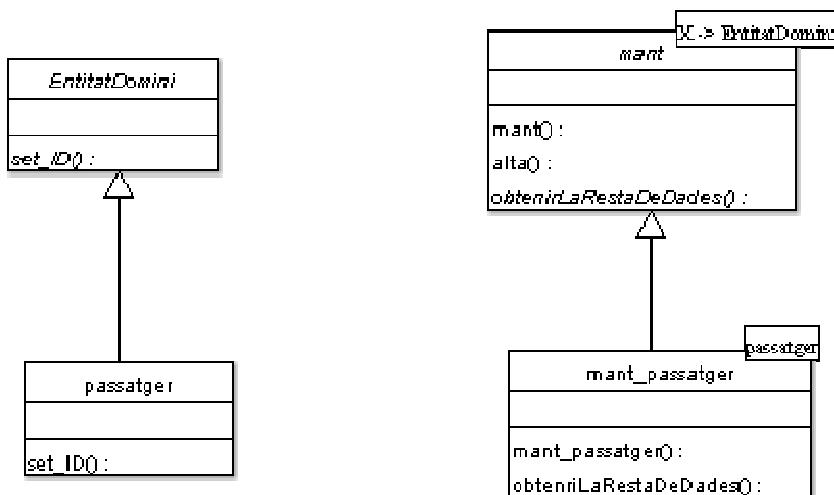
```

## 4. La potència de la generalització: un ABM genèric

Quan fem un altre ABM, per exemple, de vols o de comandes de client, gran part de la lògica de l'anterior es manté. Canvia principalment la classe d'objectes a mantenir, i per tant l'estructura dels objectes. Es tracta d'aprofitar les característiques d'un llenguatge orientat a objectes per programar un sol cop allò que és comú a tots els ABM.

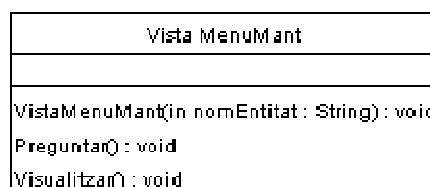
En particular, el punt crític serà que la classe d'objectes a mantenir sigui un paràmetre o argument de les classes que implementen l'ABM genèric.

- Hi haurà una classe controlador genèric per a tots els manteniments (*mant*) de qui heretaran tots els controladors de manteniment. Tindrà un paràmetre que serà la classe d'entitat del domini que ha de mantenir. Per la lògica del cas d'ús, totes les entitats del domini que s'han de mantenir han de tenir alguna mena d'identificador.



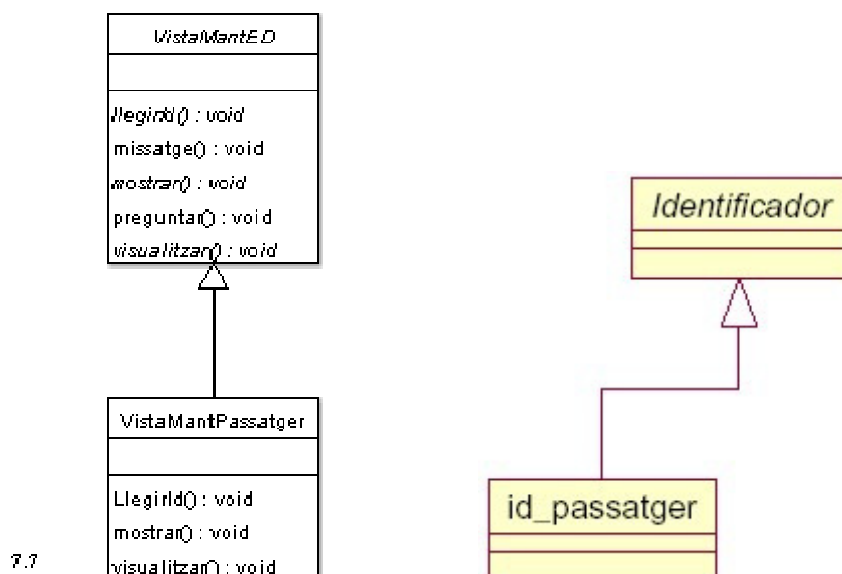
7.5

- Caldrà modificar la classe *VistaMenuMant* perquè puguem passar-li un text o literal (*string*) amb el nom de la classe d'objectes del model (subclasse d'*EntitatDomini*) que volem mantenir en cada cas.



7.6

- També caldrà una classe abstracta que generalitzi VistaMantPassatger de Passatger a una entitat de domini qualsevol. Com que les dades que cal llegir i presentar dependran de la classe d'entitat en qüestió, VistaMantPassatger haurà d'heretar d'una classe abstracta i diferida.



- Per acabar, haurem de fer el mateix amb l'identificador, amb el controlador de manteniment de la capa de domini, i amb el gestor de persistència: generalitzar-los perquè puguem aplicar-los en altres classes d'entitats del domini.

Una possible implementació en Java 1.5 la podríem obtenir modificant el que havíem fet de la forma següent (hem marcat el que es ~~treu~~ i el que s'afegeix així perquè sigui més fàcil de seguir) :

```

public abstract class mant_passatger<T extends EntitatDomini>
{
    VistaMenuMant VMM;
    VistaMantPassatgerED VMPED;
    ControladorDominiMantPassatgerED CDMPEd;
    T ed;
    String nomEntitat;
    Identificador ID;

    public mant_passatger (VistaMantED ve, T e, ControladorDominiMantED cd,
        String nomE, Identificador ide)
    {
        char OPC;
        nomEntitat = nomE;
        VMED = ve;
        ed = e;
        CDMED = cd;
        ID = ide;
        VMM = new VistaMenuMant (nomEntitat);
        VMM.visualitzar ();
        OPC = VMM.preguntar ("Quina opció vols?", '1', '2', '3', '4', '0');
        while (OPC != '0')
        {
            switch (OPC)
            {
                case '1': alta ();
                    break;
                case '2': baixa ();
                    break;
                case '3': modifica ();
                    break;
            }
        }
    }
}
  
```

```

        case '4': consulta ();
                break;
    } // switch
    VMM.visualitzar ();
    OPC = VMM.preguntar ("Quina opció vols?", '1', '2', '3', '4', '0');
} // while
} // mant_passatger ()

private protected void alta ()
{
    char OPC;
id_pass ID;
passatger PS;
VMP = new VistaMantPassatger ();
CDMPS = new ControladorDominiMantPassatger ();
    VMED.visualitzar ();
    ID = VMED.LlegirId (); // l'obté i el crea (pot validar la sintaxi d'ID)
    while (ID != null)
    {
        if (CDMPSED.existent(ID)) // CDMPSED usarà el gestor de dades
            VMED.missatge (nomEntitat + "passatger existent");
        else
        {
PS = new passatger ();
            if (PSed.set_ID (ID))
            {
                obtenirLaRestaDeDadesPassatger (ed); // obté i valida PS ed
                VMED.mostrar (PSed);
                OPC = VMED.preguntar ("Confirma (S/N): ", 'S', 'N');
                if (OPC = 'S')
                    CDMPSED.confirmat (PSed); // Comunica la validació de l'usu-
                                                    // ari, ja es pot afegir, CDMPSED
                                                    // usa el gestor de dades si convé
            }
            else
                VMED.missatge ("codi de   + nomEntitat + "passatger no vàlid");
            // if
        } // if
        VMED.visualitzar ();
        ID = VMED.LlegirId (); // l'obté i crea (pot validar la sintaxi d'ID)
    } // while
} // alta()

...
abstract void obtenirLaRestaDeDades (T ed1)
{}
...
} // mant_passatger

```

És a dir, que quedaria:

```

public abstract class mant<T extends EntitatDomini>
{
    VistaMenuMant VMM;
    VistaMantED VMED;
    ControladorDominiMantED CDMED;
    T ed;
    String nomEntitat;
    Indentificador ID;

```

```

public mant (VistaManted ve, T e, ControladorDominiManted cd,
            String nomE, Indentificador ide)
{
    char OPC;
    nomEntitat = nomE;
    VMED = ve;
    ed = e;
    CDMED = cd;
    ID = ide;
    VMM = new VistaMenuMant (nomEntitat);
    VMM.visualitzar ();
    OPC = VMM.preguntar ("Quina opció vols?", '1', '2', '3', '4', '0');
    while (OPC != '0')
    {
        switch (OPC)
        {
            case '1': alta ();
                       break;
            case '2': baixa ();
                       break;
            case '3': modifica ();
                       break;
            case '4': consulta ();
                       break;
        } // switch
        VMM.visualitzar ();
        OPC = VMM.preguntar ("Quina opció vols?", '1', '2', '3', '4', '0');
    } // while
} // mant ()

protected void alta ()
{
    char OPC;
    VMED.visualitzar ();
    ID = VMED.LlegirId ();    // l'obté i el crea (pot validar la sintaxi d'ID)
    while (ID != null)
    {
        if (CDMED.existent(ID))           // CDMED usarà el gestor de dades
            VMED.missatge (nomEntitat + " existent");
        else
        {
            if (ed.set_ID (ID))
            {
                obtenirLaRestaDeDades(ed);           // obté i valida ed
                VMED.mostrar (ed);
                OPC = VMED.preguntar ("Confirma (S/N): ", 'S', 'N');
                if (OPC == 'S')
                    CDMED.confirmit (ed);           // Comunica la validació de l'usua-
                                                    // ri ja es pot afegir, CDMED
                                                    // usa el gestor de dades si convé
            }
            else
                VMED.missatge ("codi de " + nomEntitat + " no vàlid");
            // if
        } // if
        VMED.visualitzar ();
        ID = VMED.LlegirId ();           // Valida la sintaxi d'ID
    } // while
} // alta()
...
abstract void obtenirLaRestaDeDades (T ed1)
{
}
...
} // mant

```

Llavors el controlador de manteniment del passatger seria:

```
public class mant_passatger extends mant<passatger>
{
    public mant_passatger ()
    {
        super (new VistaMantPassatger (), new passatger (),
              new ControladorDominiMantPassatger (), "passatger",
              new id_pass ());
    } // mant_passatger ()

    ...
} // mant_passatger
```

## 5. Un pas més de generalització: vistes amb camps dinàmics

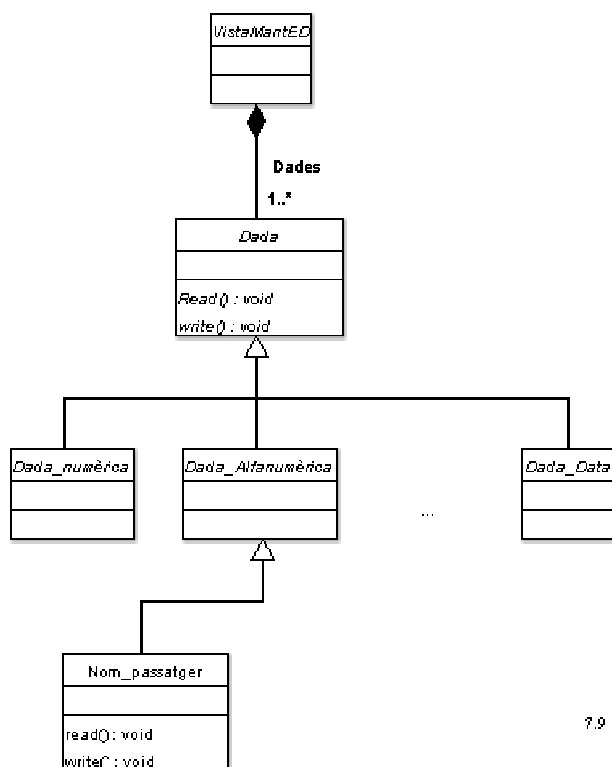
Una vista com ara `VistaMantED` serà diferida perquè hi ha dues parts de les seves funcionalitats que depenen de l'entitat a mantenir (paràmetre genèric): els camps que apareixen en pantalla i les dades a retornar al domini mitjançant les operacions `read_tribut`.

De fet, quan es concreti `VistaMantED` per a una entitat de domini particular (com ara `Passatger`), el que caldrà fer per a cada camp a mantenir és:

1. Afegir una línia al servei encarregat de presentar el formulari que presenti en pantalla el nom del camp i (en cas de Modificacions) el seu valor actual.
2. Afegir una línia al servei encarregat de llegir les entrades de l'usuari per llegir el camp.
3. Definir un servei `read_tribut` perquè la funcionalitat `obtenirLaRestaDeDades` pugui obtenir el valor llegit.

Això fa tediós modificar una vista, perquè qualsevol canvi en el conjunt de camps de l'entitat obliga a canviar com a mínim tres punts de la classe. Presentem a continuació una solució perquè un canvi en els camp a mantenir es faci en un sol punt de la classe vista.

Cada camp que ha de mantenir la vista es representa mitjançant un objecte de la classe `Dada`. Típicament, l'agregació de `Dada` a `VistaMantED` se sol implementar amb una taula (*array*) ja que així es facilita la manipulació de les dades. Els diferents tipus de Dades es poden mantenir mitjançant herència.

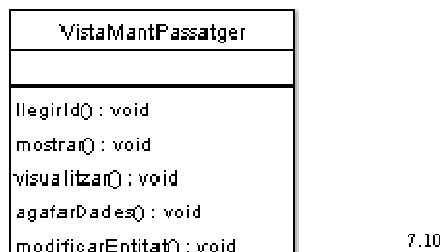


7.9

El codi de la vista se simplifica perquè

1. Presentar un formulari és reduïx a recórrer (amb una iteració) la llista de Dades agregades i presentar-les en pantalla.
2. Llegir les dades d'un formulari es fa igualment amb una iteració.
3. El pas de dades entre la Vista i el Domini es reduïx al pas d'un objecte de l'entitat del domini.

En aquest cas, el servei `obtenirLaRestaDeDades` també s'implementaria així:



```
private void obtenirLaRestaDeDades (EntitatDomini ed)
{
    VMED.agafarDades (ed);           // Ha d'estar diferida (abstract) a VistaMantED
} // obtenirLaRestaDeDades ()
```

El mètode `agafarDades` és una iteració que va agafant cada dada de `VMED` i va fent els `sets` a la entitat `ed`, cosa que inclou la validació i per tant cal tractar les excepcions i notificar-les a l'usuari. En aquest cas quan es fan modificacions o baixes la seqüència ha de ser diferent:

```
ID = VMED.llegirId ();
ed = CDMED.recuperar (ID);
VMED.mostrar (ed);
VMED.modificaEntitat (ed);
```

En aquestes darreres propostes només s'expressa un esquema de comportament, no és tracta de l'algorisme detallat, per això no hem especificat tots els paràmetres necessaris i les seves implicacions ni s'ha comprovat que `ID` correspon a un passatger existent.

## 6. Alguns principis

- Els controladors de la capa de presentació estan lligats als casos d'ús, com a màxim n'hi ha un per cas. S'intenta agrupar casos d'ús d'una "família" en un controlador. Per exemple, els ABM (alta, baixa i modificació) d'una classe.
- Els gestors de persistència (GDP) o gestors de disc estan lligats a les classes del model persistents. Normalment n'hi haurà un per classe persistent.
- Les vistes estan lligades o bé als menús o bé als casos d'ús. S'intenta aprofitar les vistes per a més d'un cas d'ús.
- Un controlador de la capa de domini integrarà tantes classes del model (amb els GDP corresponents) com li calgui per implementar els seus casos d'ús.
- A més, convé generalitzar sempre característiques comunes de les classes. Per exemple, el gestor de persistència concret de cada classe pot heretar d'un gestor abstracte totes les característiques comunes a tots els gestor de persistència.
- A la capa de presentació, com a molt, es controlarà la validesa sintàctica de les dades que introdueixi l'usuari.

## 7. Els controladors de la capa de domini

La comesa dels controladors de domini és descarregar les classes del model d'aquelles responsabilitats (mètodes o atributs) que necessiten específicament per aquest programa i que, per tant, no formen part de la seva naturalesa. Es volen treure de les classes del model perquè en dificultarien la reutilització i no es poden passar als controladors de presentació perquè tenen a veure exclusivament amb transformacions del domini i no amb el diàleg amb l'usuari. Aquestes responsabilitats normalment són els mètodes necessaris per implementar el casos d'ús, a més de la pròpia classe gestor de dades si és persistent. L'agregació a la classe del model (capa de domini) de la classe que en gestiona la seva persistència (capa de gestió de dades) la carregaria, a més, amb característiques tecnològiques específiques de l'entorn d'aquest programa.

Seguint amb el mateix programa de gestió d'una agència de viatges podríem plantejar la implementació de la reserva i venda d'un bitllet d'avió. Com a mínim hi hauria involucrades les classes del model `passatger`, `vol`, i `bitllet`. Un esquema de la implementació d'aquest cas d'ús, si més no amb els elements més importants, podria ser:

