

Tema 8

Programació Orientada a Objectes.

Classes amb atributs.

1. Introducció a la POO	2
2. Classes i objectes en Java	4
2.1 Definició de la classe	4
2.2 Definició dels objectes	6
2.3 Ús dels atributs dels objectes	7
2.4 Ús dels propis objectes	9
3. Classes niuades	15
4. Vectors d'objectes	22

1. Introducció a la POO

Un món orientat a objectes

Tot el que ens envolta són objectes:



I, com podem veure, cada objecte és d'una classe en concret:

Classe Alumne



Objectes de la classe Alumne



Classe Ordinador



Objectes de la classe Ordinador



Classe Casa



Objectes de la classe Casa



Què és la Programació Orientada a Objectes (POO)?

És una forma de programar que es basa precisament en això: en els objectes. Primer definirem una classe i, a partir d'ella, podrem crear els objectes d'eixa classe (igual com creem variables d'un tipus determinat).

Què es posa en una classe?

En una classe posarem allò que volem que descriga els objectes d'eixa classe. Si ens adonem, podem descriure un objecte per "com és" i "què pot fer":

- Com és l'objecte? Voldrem indicar unes característiques o **atributs**.
- Què pot fer l'objecte? Voldrem indicar unes operacions o **mètodes**.

Per a anar pas a pas, en este tema només vorem els atributs de les classes. Els mètodes de les classes ho vorem al tema següent.

Exemple gràfic d'una classe i els seus objectes

Suposem que volem fer un programa sobre la gestió dels alumnes de l'institut.

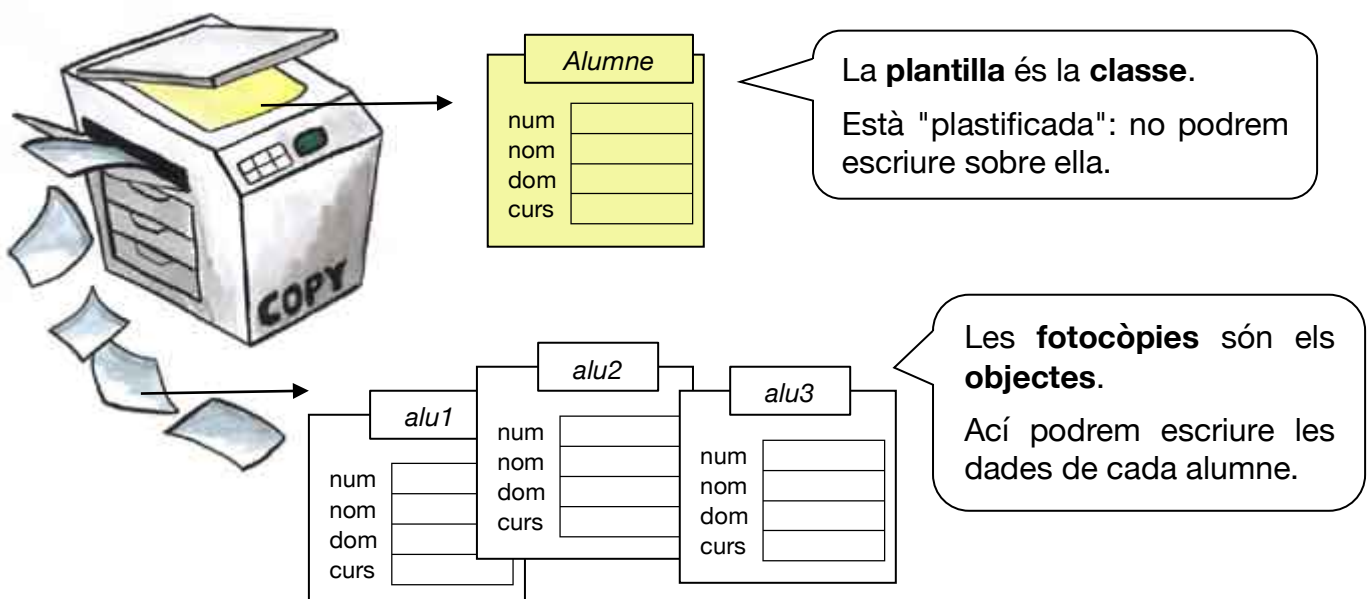
Cada alumne és un objecte.

Atributs de la classe *Alumne*.

- Com és cada alumne? Voldríem saber el seu codi, nom, domicili i curs.
- Què pot fer cada alumne? Matricular-se, examinar-se...

Mètodes de la classe *Alumne*.

Si l'institut no estiguera informatitzat, per a guardar les dades de cada alumne caldrien fitxes amb la mateixa estructura (codi, nom, domicili i curs). Per tant, necessitaríem una **plantilla** on estiguera eixa estructura i fer **fotocòpies** d'ella:



Terminologia de la POO

CONCEPTES POO	TERMINOLOGIA INFORMAL	ANALOGIA EN LA VIDA REAL	EXEMPLES
Classe	Tipus	Plantilla plastificada	<i>Alumne</i>
Objectes	Variables	Fotocòpies a emplenar	<i>alu1, alu2, alu3</i>
Atributs	Variables membre	Apartats de la fitxa	<i>num, nom, dom, curs</i>
Mètodes	Funcions i proced.	Accions sobre la fotocòpia	<i>matricularAlumne(), canviDeCurs()...</i>

Més endavant vorem altres conceptes relacionats amb la POO, com herència, abstracció, polimorfisme, encapsulament, acoblament i cohesió.

2. Classes i objectes en Java

Com ja hem dit, per a poder declarar una **objecte**, primer necessitem crear una **classe** (plantilla). Després ja podrem definir els nostres objectes en base a eixa classe.



2.1 Definició de la classe

Anem a veure com podem definir una classe en Java. Però com ja hem dit, en este tema només vorem una classe com un conjunt d'atributs. En el tema següent dins la classe posarem també les operacions o mètodes que es poden fer en cada objecte.

Exemple:

```
class Alumne {  
    int num;  
    String nom;  
    String cicle;  
    int curs;  
}
```

La classe *Alumne* és la plantilla a partir de la qual crearem objectes on guardarem les dades de cada alumne.

Sintaxi:

```
class NomClasse {  
    [public | private] tipus camp1;  
    ...  
    [public | private] tipus campN;  
}
```

Ja vorem si hem de posar que un atribut és *public* o *private*.

De moment crearem les classes en el mateix fitxer on tenim el Main, fora de la classe principal, però ja vorem que convindria que cada classe anara en un fitxer (que tindria el mateix nom de la classe).

```
class Alumne {  
    int    num;  
    String nom;  
    String cicle;  
    float  curs;  
}  
  
class ProjecteMeu {  
    public static void main(String[] args){  
        ...  
    }  
}
```

ProjecteMeu.java

Exercicis de definir classes en Java

1. Crea un projecte de nom Proves i defineix en ell les classes corresponents a les següents estructures de dades:
 1. Una data: dia, mes i any
 2. Un temps: hores, minuts, segons i centèsimes
 3. Un rectangle: cantó superior dreta (x1, y1) i cantó inferior esquerra (x2, y2)
 4. Un concursant: nom complet, nom artístic i any de naixement
 5. Un CD d'àudio: grup, títol del disc, any publicació, quantitat de cançons
 6. Un nom complet de persona: nom, primer cognom i segon cognom
 7. Un número de telèfon fix: prefix i resta del número
 8. Un domicili: carrer, número, pis, porta, codi postal, població i comarca
 9. Un color RGB: format per 3 enters (quantitat de roig, de verd i de blau).

2.2 Definició dels objectes

Abans hem vist com crear en Java la **classe** ("plantilla") *Alumne*. Ara volem guardar les dades de cada alumne. Per a això crearem **objectes** ("fotocòpies") a partir d'eixa classe.

En Java, esta definició d'objectes es fa amb la paraula **new** (la que usem per als vectors, ja que també són objectes) i indicant la classe a partir de la qual es creen.

2 FORMES DE CREAR OBJECTES		
	1r) Definim l'objecte 2n) Li reservem memòria	Definim l'objecte i li reservem memòria en una sola instrucció.
Sintaxi	NomClas nomObj ; ... nomObj = new NomClas();	NomClas nomObj = new nomClas();
Exemple	Alumne alu1 , alu2 ; ... alu1 = new Alumne(); alu2 = new Alumne();	Alumne alu1 = new Alumne(); Alumne alu2 = new Alumne();

Es com si haguérem definit les "variables" *alu1* i *alu2* de "tipus" *Alumne*.

Exercici de definir objectes en Java

2. En el *Main* del projecte *Proves* crea objectes de les classes que has definit abans. Practica amb les 2 formes vistes.

2.3 Ús dels atributs dels objectes

Ja hem creat la classe (el nou "tipus") i els objectes ("variables" d'eixe nou tipus). Ara anem a utilitzar eixos objectes.

Per a accedir als atributs dels objectes cal indicar el nom de l'objecte, un punt i el nom de l'atribut.

nomObjecte.nomAtribut

Eixos atributs podem usar-los com qualsevol altra variable.

Exemples d'usos d'atributs:

```
...
System.out.println("INTRODUEIX DADES DE L'ALUMNE:");
System.out.println("Número: "); alu1.num = llegirEnter();
System.out.println("Nom: ");    alu1.nom = llegirCadena();
System.out.println("Curs: ");   alu1.curs = llegirReal();

alu1.cicle = "DAM"

...

if (alu1.curs == 1 && totAprovat) {
    System.out.println("L'alumne " + alu1.nom + " passa a 2n");
    alu1.curs++;
}
```

Assignació per teclat

Assignació directa

Consulta

Concatenació, impressió..

Autoincrement

Veiem que els atributs dels objectes s'utilitzen igual que les variables "normals": assignar-los un valor, consultar-lo, mostrar-lo per pantalla...

Exemple complet de definició de classe, objectes i ús dels atributs:

```
import java.io.*;
```

```
//***** CLASSE Alumne *****
```

```
class Alumne {  
    String nom;  
    String cognom;  
    String [] telefon = new String [3]; // un vector de 3 possibles telèfons  
    int edat;  
}
```

Definirem la classe fora de la classe principal (encara que podria anar dins, però ja ho vorem).

Sol anar en un altre fitxer dedicat a la classe.

```
//***** CLASSE PÚBLICA PRINCIPAL *****
```

```
public class NomPrograma {
```

```
    public static void main(String[] args) {
```

```
        // ----- Definició d'objectes de la classe que hem creat ---
```

```
        Alumne a1 = new Alumne();
```

```
        Alumne a2 = new Alumne();
```

Crearem els objectes dins del *main* (o de qualsevol funció o procediment).

```
        // ----- Ús dels atributs dels objectes -----
```

```
        a1.nom = "Miquel Josep";
```

```
        a1.cognom = "Garcia";
```

```
        a1.edat = 17;
```

```
        a1.telefon[0] = "961712222";
```

```
        a1.telefon[1] = "961702299";
```

I també ací usarem eixos objectes.

```
        System.out.println(a1.nom + " " + a1.cognom);
```

```
        for (int i=0; i<a1.telefon.length; i++) {
```

```
            System.out.println(a1.telefon[i]);
```

```
        }
```

```
    }
```

```
}
```

Exercici sobre l'ús dels atributs dels objectes

3. En el projecte *Proves* assigna valors als objectes que haves creat i mostra el seu resultat per pantalla.

2.4 Ús dels propis objectes

Ja hem vist com treballar amb els atributs dels objectes. Ara vorem com tractar els objectes en conjunt (sense indicar els atributs).

Quan usem un objecte (sense indicar cap atribut), estarem accedint a l'**adreça de memòria** on estan les dades de l'objecte, igual que passa en els arrays.

Exemple:

```
class Alumne {
    int num;
    int edat;
    int curs;
}

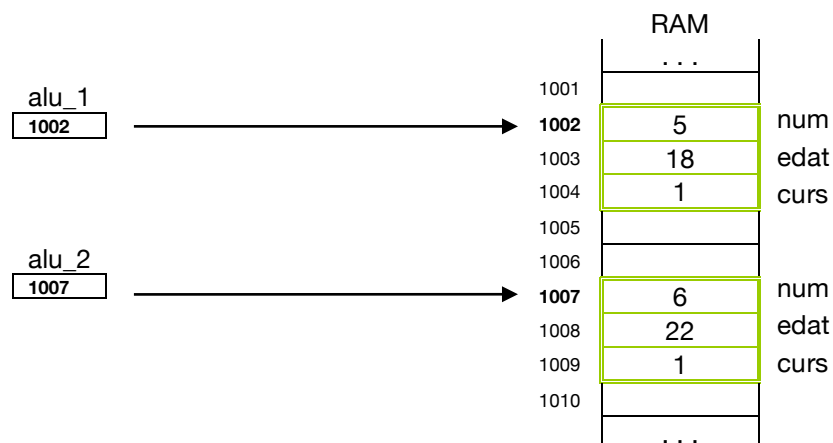
class ProgramaPrincipal {
    public static void main(String [] args){

        Alumne alu_1 = new Alumne();
        Alumne alu_2 = new Alumne();

        alu_1.num = 5;    alu_1.edat = 18;    alu_1.curs = 1;
        alu_2.num = 6;    alu_2.edat = 22;    alu_2.curs = 1;
        ...
    }
}
```

Imaginem que se li assigna l'adreça 1002.

Imaginem que se li assigna l'adreça 1007.



Per tant, hem d'anar amb compte quan fem coses com:

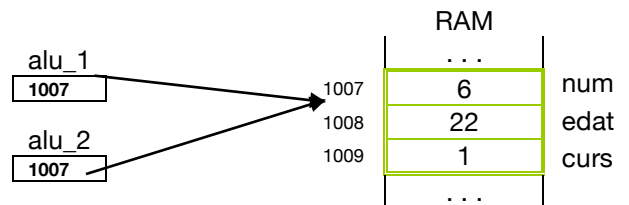
```
if (alu_1 == alu_2) ...
alu_1 = alu_2;
System.out.println(alu_1);
```

Segurament no estarem fent allò que pretenem ja que estarem accedint a eixe 1002 i 1007. Ara ho vorem.

a) Copiar objectes

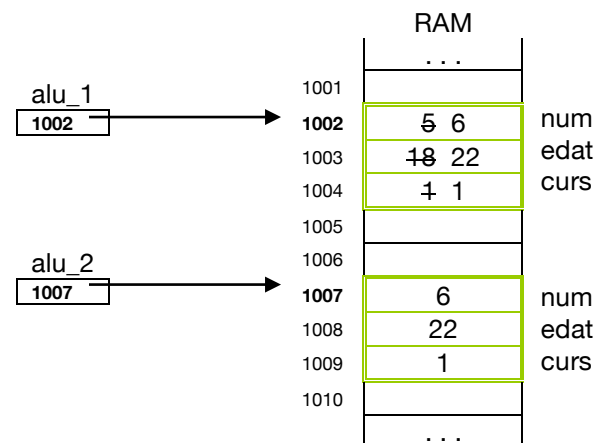
Què passa si copiem objectes amb `alu_1 = alu_2;` ?

Problema: Apuntaran a la mateixa zona.
Per tant, si després modifiquem algun atribut d'un dels objectes, l'altre també tindrà eixes modificacions, ja que els dos objectes estan apuntat a la mateixa zona de memòria on estan els atributs.



Solució : si el que volíem fer és una còpia dels valors de l'objecte (i no que apunten al mateix objecte), cal copiar un a un els atributs:

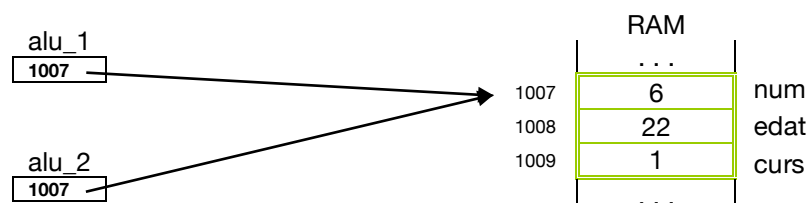
```
alu1.num = alu2.num;  
alu1.edat = alu2.edat;  
alu1.curs = alu2.curs;
```



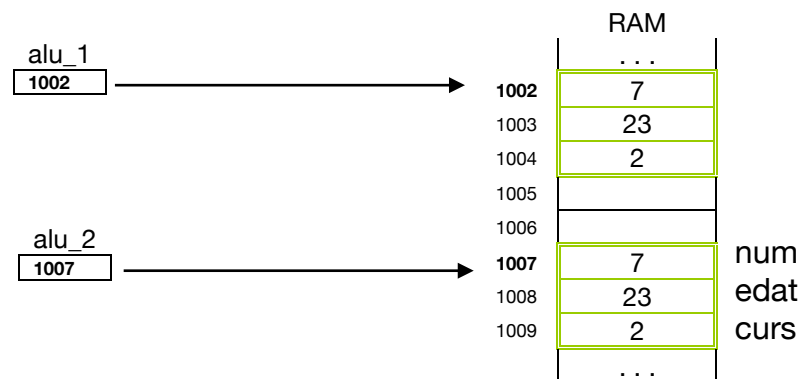
b) Comparar objectes

Què passa si comparem objectes amb `(alu_1 == alu2)` ?

- Només serà **true** si els 2 objectes apunten a la mateixa zona de memòria:



- Però si apunten a zones diferents, serà **false**, encara que els valors dels atributs dels objectes siguin els mateixos:



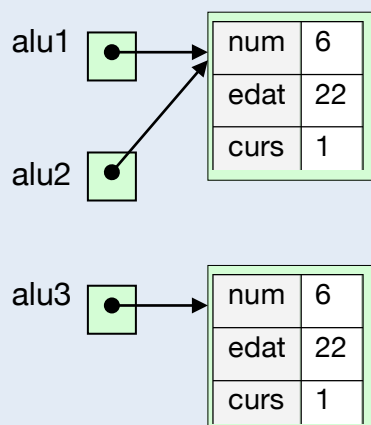
Solució : si el que volíem fer és comparar els valors dels 2 objectes (i no comparar si apunten al mateix objecte), cal comparar cadascun dels atributs:

`¿ (alu1.num == alu2.num) && (alu1.edat == alu2.edat) && (alu1.curs == alu2.curs) ?`

Representació gràfica dels objectes

Per a recalcar que un objecte no guarda directament els valors dels seus atributs sinó una adreça a elles, a partir d'ara ho representarem gràficament amb una fletxa.

Per exemple, tenim 3 objectes de tipus *Alumne*. Imaginem que en un moment donat tenim esta situació:



Segons eixa situació, observem que:

- ✓ `alu1 == alu2`
- ✓ `alu1 != alu3`
- ✓ `alu2 != alu3`
- ✓ Si modifique `alu2.edat`, **també** estic modificant `alu1.edat`
- ✓ Si modifique `alu2.edat`, **no** estic modificant `alu3.edat`

c) Passar un objecte a una funció

Podem passar un objecte com a paràmetre a una funció. I estarem passant-lo per referència (no per valor). Com ja vam veure, això significa que, els canvis produïts en l'objecte dins la funció es mantenen fora de la funció.

```
class Alumne {
    int num;
    int edat;
    int curs;
}

class ProjectePrincipal {
    public static void main (String[] args) {
        Alumne alu_1 = new Alumne();

        // Passem l'adreça de l'objecte buit a la funció, i la funció l'omplirà.
        llegirAlumne(alu_1);

        // Passem l'adreça de l'objecte amb valors per a que s'imprimisquen.
        imprimirAlumne(alu_1);
        ...
    }

    public static void llegirAlumne(Alumne a) {
        // Llegim de teclat 3 valors i ho posem a l'objecte:
        imprimir("Dóna'm el número de l'alumne: ");
        imprimir("Quants anys té? ");
        imprimir("A quin curs va? ");
        a.num = llegirEnter();
        a.edat = llegirEnter();
        a.curs = llegirEnter();
    }

    public static void imprimirAlumne(Alumne a) {
        // Imprimim els valors que té l'objecte
        System.out.println("-- DADES ALUMNE --");
        System.out.println("Número: " + a.num);
        System.out.println("Edat: " + a.edat);
        System.out.println("Curs: " + a.curs);
    }
}
```

num	
edat	
curs	

Exercicis de passar objectes a procediments

4. En el projecte *Proves* crea les funcions *llegirCD* i *imprimirCD* semblants a l'exemple anterior, passant un CD com a paràmetre. Crida-les des del main.



d) Retornar un objecte

Una funció també pot retornar un objecte. Veiem un exemple on tenim dos funcions que retornen un objecte. Una que no li passes cap paràmetre i altra que sí.

```
class Alumne {
    int num;
    int edat;
    int curs;
}

class ProjectePrincipal {



    public static void main (String[] args) {

        Alumne alu_1;  

No reservem memòria per a alu_1 ja que s'encarregarà la funció llegirAlumne().





        // No li passem res a la funció.
        // La funció crearà un alumne, li assignarà valors i ens el retornarà:
        alu_1 = llegirAlumne();


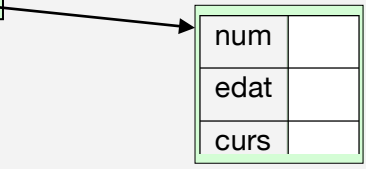
        Alumne alu_2;  

No reservem memòria per a alu_2 ja que s'encarregarà la funció clonarAlumne().



        // Li passem un alumne a la funció.
        // La funció en crearà un altre (amb els mateixos valors) i ens el retornarà:
        alu_2 = clonarAlumne(alu_1);
        ...
    }

    public static Alumne llegirAlumne() {
        Alumne a = new Alumne();  
        imprimir("Dóna'm el número de l'alumne: ");
        imprimir("Quants anys té? ");
        imprimir("A quin curs va? ");
        a.num = llegirEnter();
        a.edat = llegirEnter();
        a.curs = llegirEnter();
        return a;
    }

    public static Alumne clonarAlumne(Alumne a) {
        Alumne aCopiat = new Alumne();  
        aCopiat.num = a.num;
        aCopiat.edat = a.edat;
        aCopiat.curs = a.curs;
        return aCopiat;
    }
}
```

num	
edat	
curs	

num	
edat	
curs	

Exercicis de retornar objectes

Modifica el projecte *proves*:

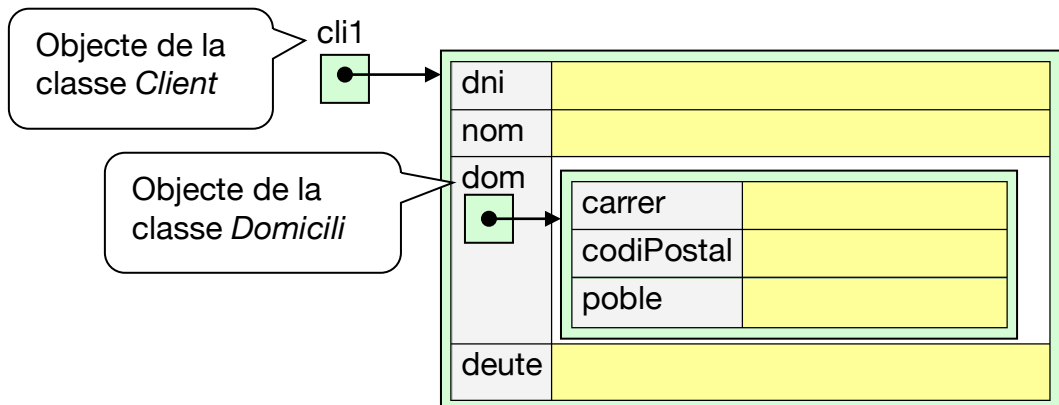
5. Procediment **incrementaAny**, a la qual li passes una data (de la classe Data) i t'incrementa l'any en 1 (però no mostra res). És a dir, cal fer el següent:
 - a) Fes el procediment al qual li passes com a paràmetre una data (dis-li de nom: data). Incrementa en 1 l'any.
 - b) En el main crea una data (dis-li de nom: d1), posa-li valors i crida al procediment incrementaAny passant-li eixa data com a paràmetre.
 - c) Comprova en el main que s'ha modificat l'any de d1 (mostra-ho per pantalla).
6. Funció incrementaAnyEnDataNova, a la qual li passes una data i te'n retorna una altra igual però amb un any més. És a dir, cal fer el següent:
 - a) Fes la funció, a la qual li passes com a paràmetre una data (dis-li de nom: data). La funció haurà de fer el següent:
 - Crear una altra data, de nom dNova (declarar-la i reservar-li memòria amb new).
 - Copiar en eixa altra data (dNova) les dades de la data del paràmetre però amb un any més.
 - Retornar eixa nova data.
 - b) En el main crea una data (dis-li de nom: dOrigen), posa-li valors i crida a la funció incrementaAnyEnDataNova passant-li eixa data com a paràmetre. Recorda que hauràs d'arreglar la data retornada. Per a això, caldrà crear-te prèviament una altra data (dis-li de nom: dDestí), la qual no caldrà que li reserves memòria amb el new.
 - c) Comprova en el main que la nova data (dDestí) té els valors esperats: els mateixos que dOrigen però amb un any més (mostra-ho per pantalla).

3. Classes niuades

Ja hem vist que una classe és una estructura que conté distints atributs. Eixos atributs poden ser variables (d'un cert tipus)... o objectes (d'una certa classe).

És com si tinguérem una classe dins d'altra. Eixa classe de dins es diu que és una **classe niuada**.

Exemple:



Implementació:

1r) Crearem la classe **Domicili**

```
class Domicili {  
    String carrer;  
    int    codiPostal;  
    String poble;  
}
```

Esta classe podrà estar definida dins la classe *Client* però nosaltres la posarem fora (per si volguérem crear objectes de la classe *Domicili*).

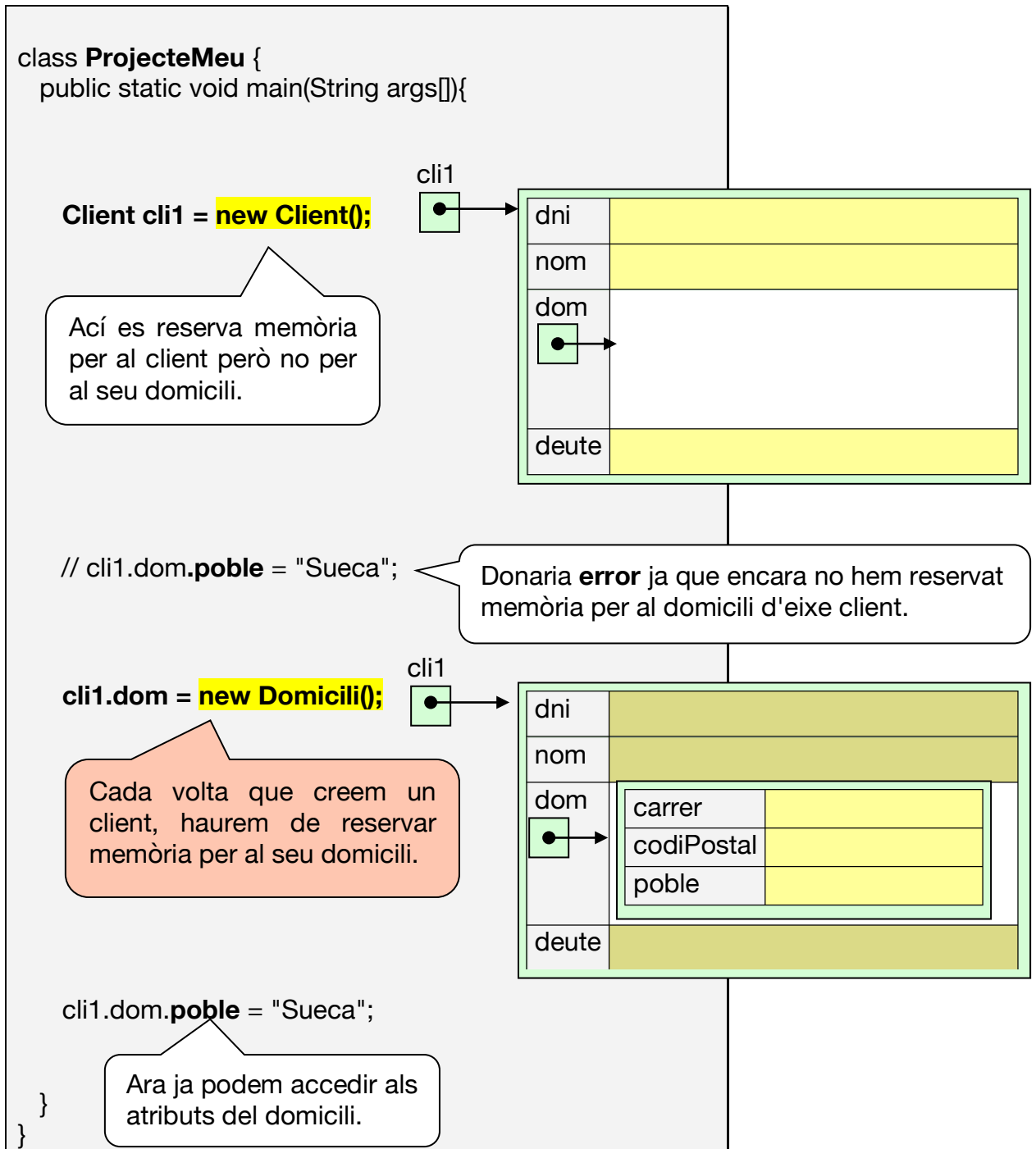
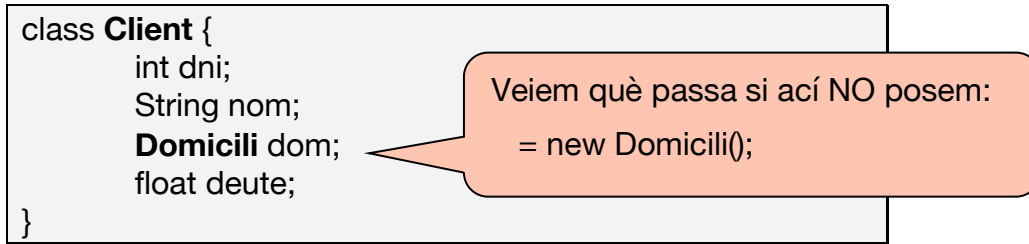
2n) Crearem la classe **Client**

```
class Client {  
    int dni;  
    String nom;  
    Domicili dom;  
    float deute;  
}
```

O bé: **Domicili dom = new Domicili();**

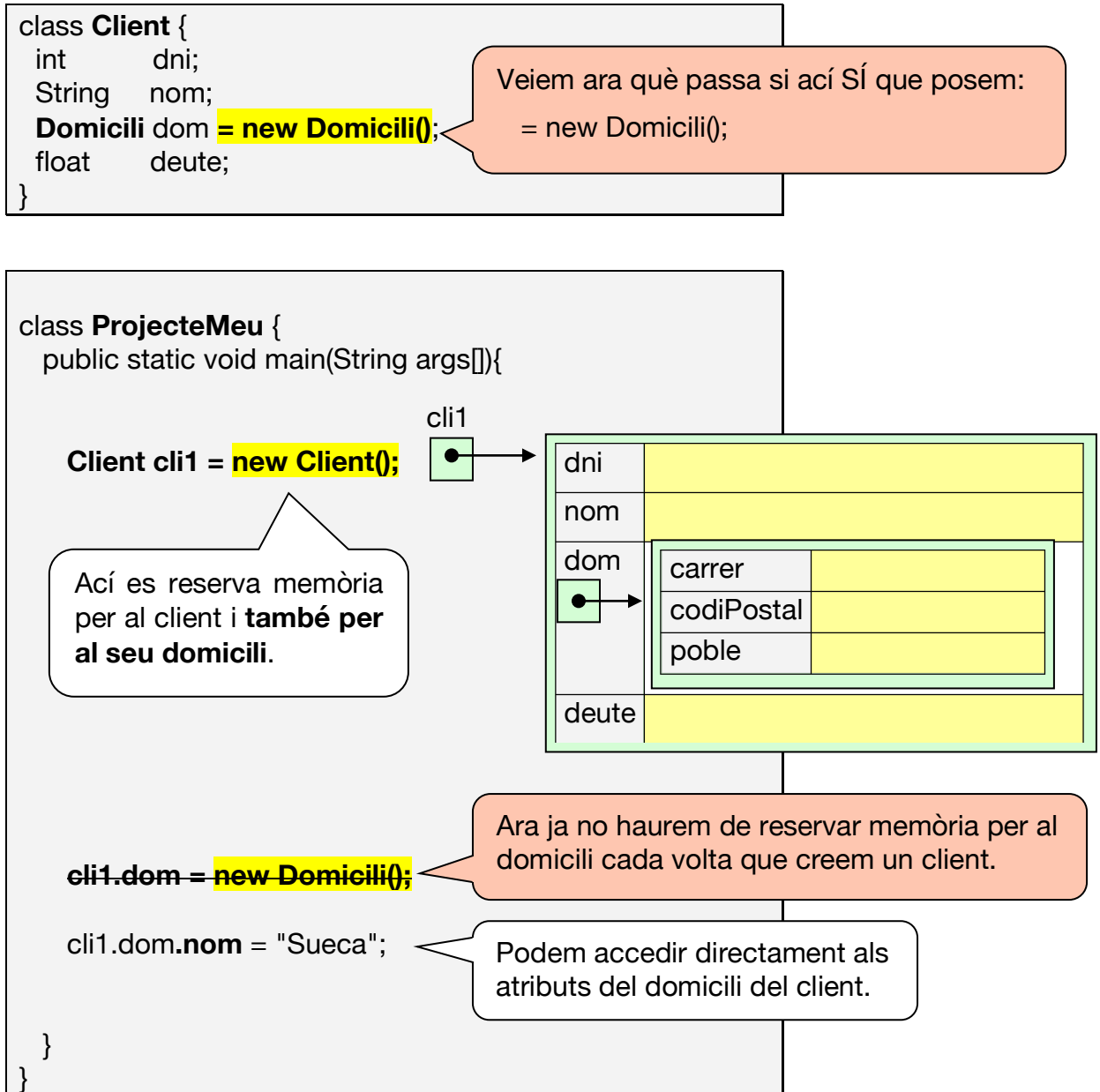
Ara vorem la diferència d'indicar o no la reserva de memòria per a la subclasse.

a) Sense instanciar l'atribut objeje (és a dir, sense *new*):



 Espai que es resreva amb el **new**.

b) Instanciant l'atribut objeje (és a dir, amb *new*):



A la vista d'això... quan definirem un atribut objeje amb el *new* i quan sense *new*?

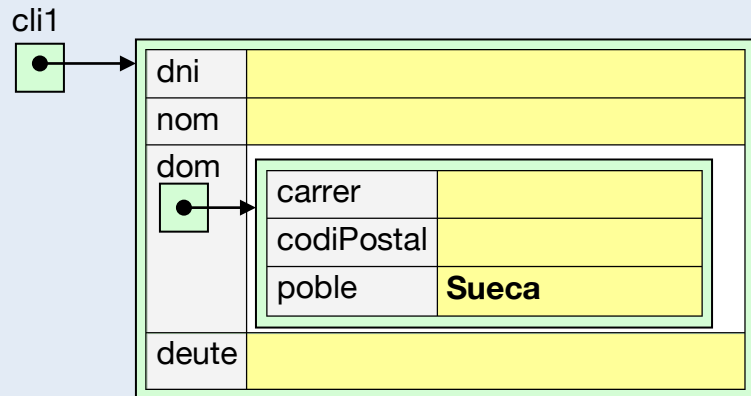
- Si anem a tindre molts objectes de la classe *Client* però només a uns pocs d'ells els posarem domicili, reservarem memòria per al domicili només als clients que els faça falta. Per tant, **sense new**.
- Si sempre que creem un client li anem a posar domicili, deixarem la classe preparada per a no reservar memòria per al domicili cada volta que creem un client, ja que ens serà més fàcil de programar. Per tant, **amb new**.

Nota: veiem que, quan usem el “punt” (.) estem fent ús de les “fletxes” (→):

cli1.dom.poble = "Sueca";

és com fer:

cli1 → dom → poble = "Sueca";



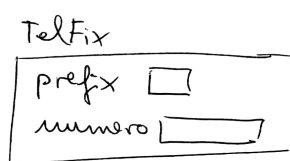
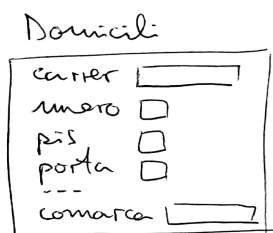
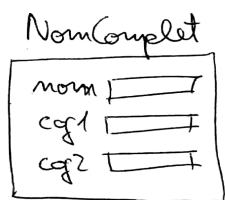
Exercicis sobre classes niuades

7. En el projecte Proves:

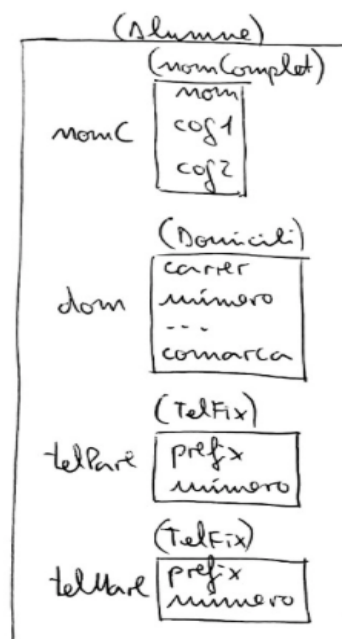
- Crea la classe Cercle amb els atributs radi, colorVora i colorDins, tenint en compte que els colors han de ser objectes de la classe Color que ja estava definida. Fes-ho de la primera forma que hem vist: la que en la definició de la classe NO es reserva memòria per a la subclasse.
- Després, en el main crea l'objecte cercle1 (de la classe Cercle) i ompli'l de dades qualsevol.
- Mostra per pantalla les dades de cercle1
- Executa el projecte per veure que funciona.

8. Crea el projecte Institut. A continuació:

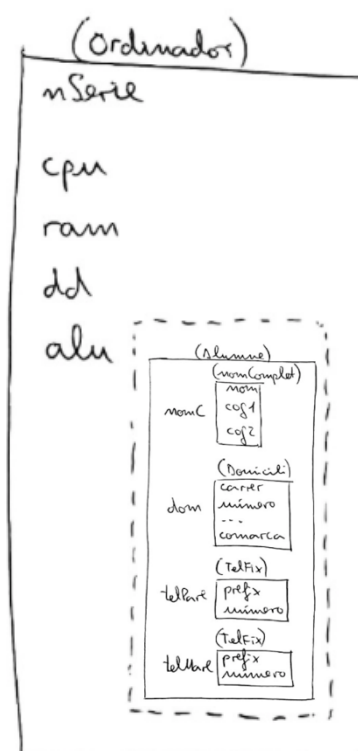
- a) Copia en ell les classes NomComplet, Domicili i TelFix (que tenies en el projecte Proves).



- b) Crea la classe Alumne que tinga 3 camps: un nom complet, un domicili, un telèfon del pare i un telèfon de la mare, basant-te en les classes que ja tens creades. És a dir: caldrà fer ús de classes niuades. Com de tots els alumnes és normal guardar eixes dades, ho farem de la segona forma que hem vist: la que en la definició de la classe Sí que es reserva memòria per a la subclasse.

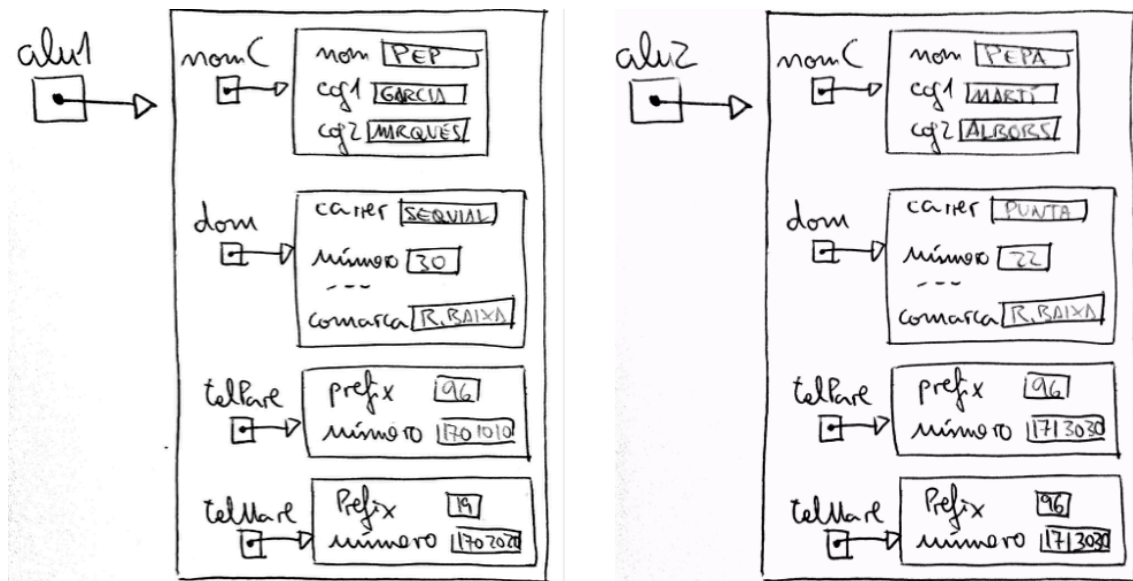


- c) Crea la classe Ordinador, sabent que de cadascun voldrem guardar el número de sèrie (enter), la cpu (String), ram (enter) i dd (enter) i, a més, qui és l'alumne que l'utilitza. L'alumne serà un objecte de la classe Alumne. És a dir: caldrà fer ús de classes niuades. Com no tots els ordinadors tindran un alumne assignat, ho farem de la primera forma que hem vist: la que en la definició de la classe NO es reserva memòria per a la subclasse.

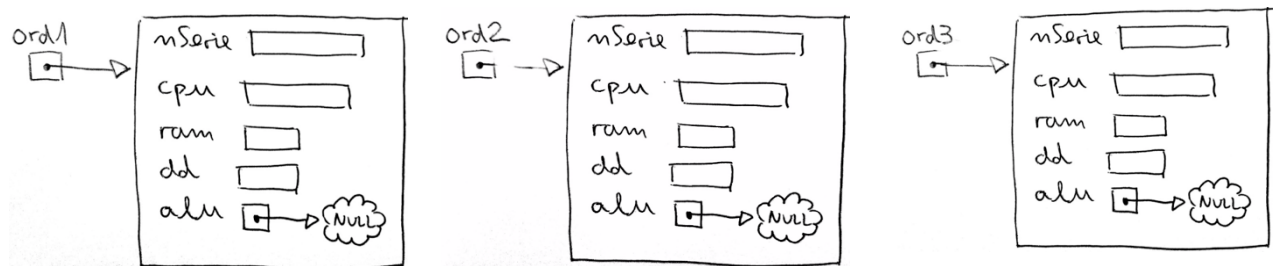


9. En el projecte Institut, en el main:

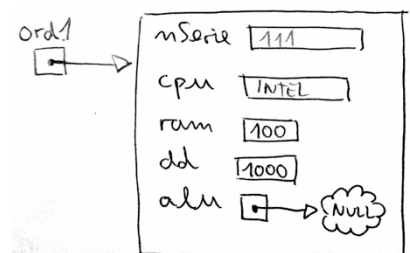
a) Crea 2 alumnes (alu1 i alu2) i omplir-los de dades qualssevol.



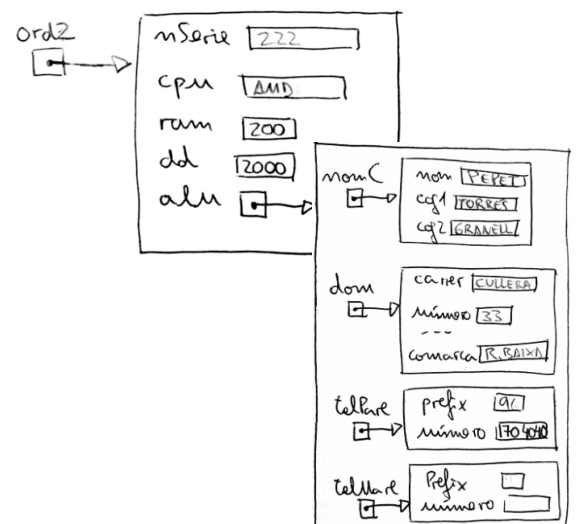
b) Crea 3 ordinadors (ord1, ord2 i ord3), sense omplir-los de dades.



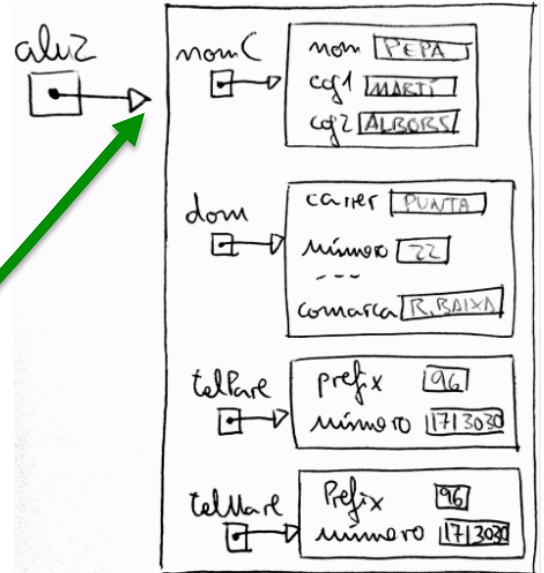
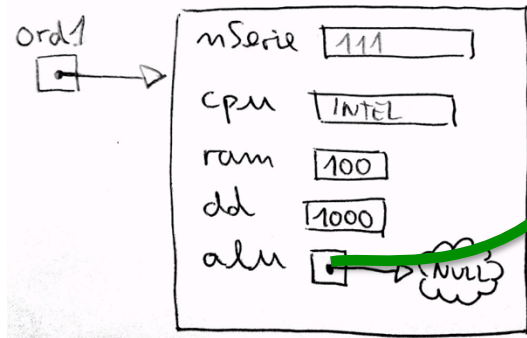
c) Ompli ord1 de dades qualssevol però sense assignar-li cap alumne.



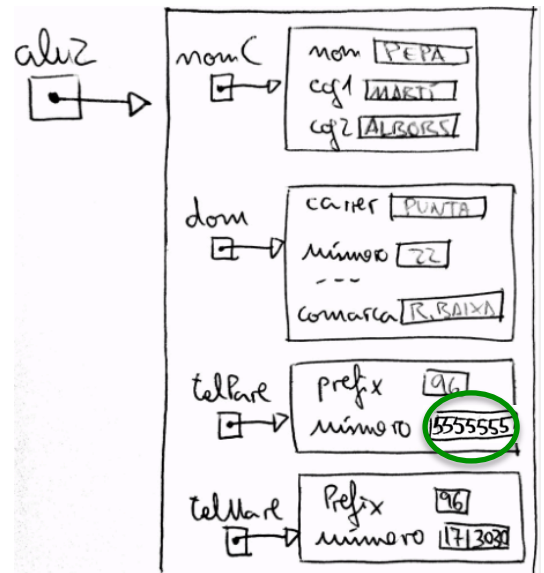
d) Ompli ord2 de dades qualssevol assignant-li les dades d'un nou alumne.



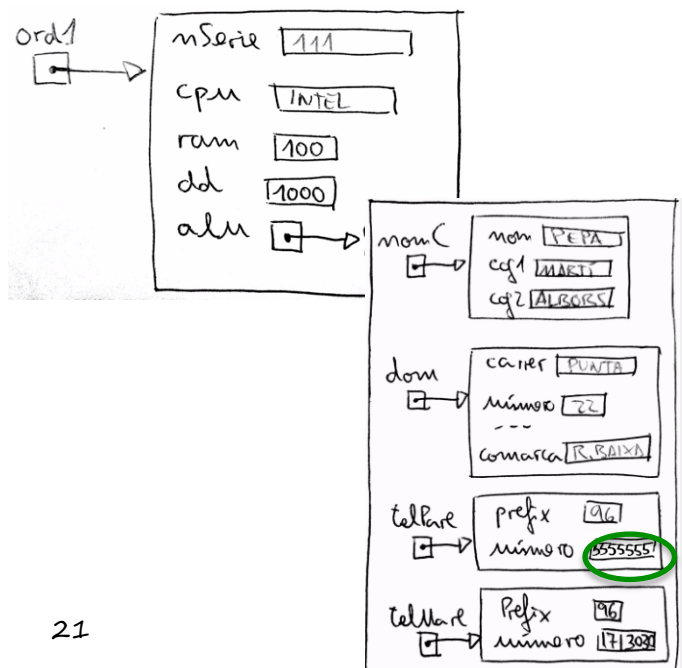
e) Assigna a l'ordinador 1 l'alumne 2.



f) Canvia el número de telèfon del pare de l'alumne 2 (per exemple: 5555555)



g) Mostra per pantalla el telèfon del pare de l'alumne que està usant l'ordinador 1. No ho facis a partir de l'alumne 2, sinó a partir de l'ordinador 1. Comprova que ha canviat (ha de mostrar 5555555).



4. Vectors d'objectes

L'ús simultani de vectors i objectes ens proporciona una eina potent per a guardar i manipular informació.

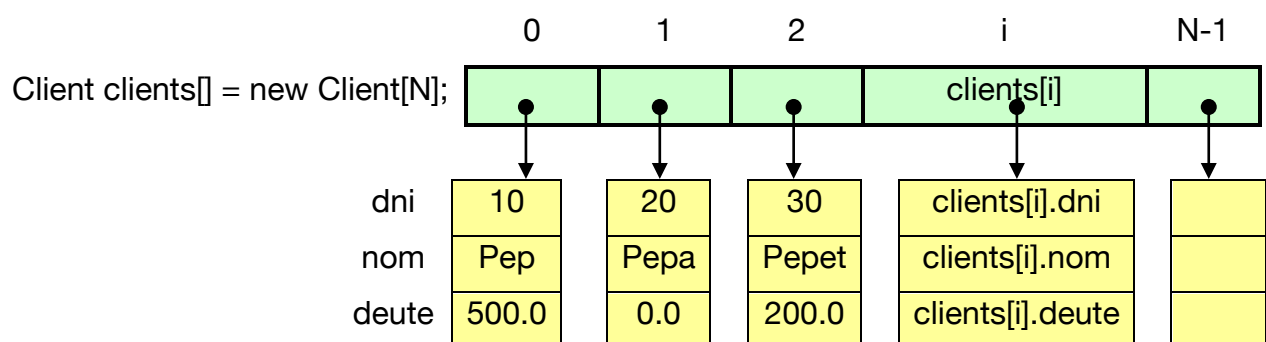
Per exemple, suposem que volem guardar les dades dels N clients d'una empresa:

	0	1	2	...	N-1
dni	10	20	30		
nom	Pep	Pepa	Pepet		
deute	500.0	0.0	200.0		

Abans de vore este tema, haguérem implementat això com 3 vectors paral·lels:

	0	1	2	...	N-1
<code>int dniClients[] = new int[N];</code>	10	20	30		
<code>String nomClients[] = new String[N];</code>	Pep	Pepa	Pepet		
<code>float deuteClients[] = new float[N];</code>	500.0	0.0	200.0		

Però ara ho implementarem com un únic vector... de clients. És a dir, d'objectes de la classe Client:



Compte! El vector és d'objectes. Recordem que un objecte és només una referència i cal reservar memòria per a eixe objecte. Per tant, el *new* del vector d'objectes només reserva espai per a les referències, i **caldrà fer un new per a cada objecte del vector**. Ara ho vorem amb la implementació d'este exemple.

Exemple: creació i ús d'un vector d'objectes (100 clients).

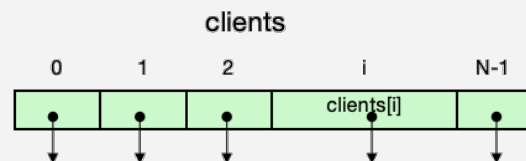
// Definim la classe Client

```
class Client {  
    int    dni;  
    String nom;  
    float  deute;  
}
```

```
class Projecte {  
    public static void main(String[] args) {  
        final int N = 100;
```

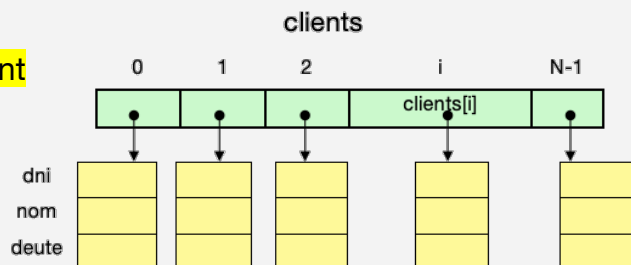
// Definim un vector de N clients

```
Client clients[] = new Client[N];
```



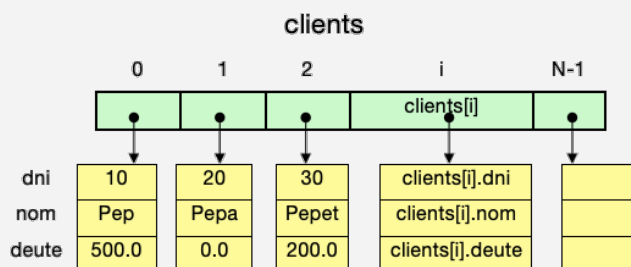
// Reservem memòria a cada client

```
for (int i=0; i<N; i++) {  
    clients[i] = new Client();  
}
```



// Omplim les dades dels clients:

```
for (int i=0; i<N; i++) {  
    imprimir("DNI: ");  
    clients[i].dni = llegirEnter();  
    imprimir("Nom: ");  
    clients[i].nom = llegirCadena();  
    imprimir("Deute: ");  
    clients[i].deute = llegirReal();  
}
```



// Mostrem les dades dels clients:

```
imprimir("DNI \t NOM \t DEUTE");  
for (int i=0; i<N; i++) {  
    imprimir(clients[i].dni + "\t" + clients[i].nom + "\t" + clients[i].deute);  
}  
}
```

Nota: també es poden crear matrius n-dimensionals d'objectes, o objectes que tenen alguna matriu com a atribut, etc.

10. Biblioteca

Una biblioteca vol emmagatzemar informació de cada llibre que té:

- Codi de referència (alfanumèric)
- Autors (màxim, 3)
- Títol
- Editorial
- Any

- a) Implementa l'estructura necessària per a guardar 100 llibres.
- b) Reserva memòria per a cadascun d'eixos llibres.
- c) Dona d'alta un llibre qualsevol.

11. Lloguer de cotxes

Un empresa de lloguer de cotxes vol tindre guardada la informació de cadascun dels cotxes:

- matrícula (lletres i número),
- marca
- model
- data de compra (usa la classe *Data* que feres)
- kms

- a) Implementa l'estructura necessària per a guardar 100 cotxes (però no reserves memòria per a als cotxes).
- b) Fes un bucle per a omplir les dades dels cotxes.
- c) Mostra les dades de tots els cotxes.

12. Empresa constructora

L'empresa constructora "MACA S.A." promou la construcció de l'edifici de luxe "Xequin-Pis". L'edifici està organitzat en 6 escales (de l'A a l'F). Per cada escala hi



ha 8 plantes, i en cada planta hi ha 5 portes. La constructora vol tindre registrat de cada vivenda els metres quadrats, habitacions i preu. A més, si la vivenda està venuda, també vol guardar el nom i NIF del nou propietari.

- a) Definix la classe **Vivenda** amb les variables: m2, q_hab, preu, nom, nif
- b) Definix **edifici** (al *main*), com una matriu tridimensional de vivendes. Utilitza les constants necessàries per a establir les dimensions de la matriu.
- c) Definix el procediment **construirVivenda**, al qual se li passa com a paràmetre l'edifici. El procediment demanarà per teclat la identificació de la vivenda i les característiques i les guardarà.
- d) Definix el procediment **comprarVivenda**, al qual se li passa com a paràmetre l'edifici. El procediment demanarà per teclat la identificació de la vivenda. Si està fabricada ($m2 > 0$) i per vendre (camp propietari buit), es demanaran les dades del propietari i es guardaran en el lloc corresponent.
- e) Definix el procediment **propietats**, al qual se li passa com a paràmetre l'edifici i el nif d'una persona, i ha de mostrar les dades de totes les seues vivendes.
- f) Crea al main l'aplicació principal amb un bucle i un menú amb les opcions:
 1. Construir vivenda
 2. Comprar vivenda
 3. Mostrar propietats d'algú
 4. Eixir

13. Taller de cotxes

En un taller de cotxes, cada volta que un treballador acaba una feina, inserix en l'ordinador el seu nom, la data (dia, mes i any) i quantes hores i minuts ha estat treballant.



- a) Definix la classe *Feina* amb les variables: *nom*, *dia*, *mes*, *any*, *hores*, *minuts*.
- b) El programa serà un bucle amb un menú:
 1. Afegir feina
 2. Llistar feines d'un treballador
 3. Eixir

Opció 1. Demanar dades d'una feina i afegir-la en una llista de feines.

- a) Si ho fas amb un vector de feines, caldrà tindre una variable on indicar quantes feines tenim en cada moment, per a saber en quin lloc del vector posarem la nova feina.
- b) Però seria millor un ArrayList de feines ja que realment és una llista perquè va variant la grandària al llarg del programa.

Opció 2. Demanar el nom del treballador. Es mostrarà per pantalla una línia per cada feina seua (amb les dades corresponents) i, a la dreta, l'import a cobrar corresponent, a 40 € l'hora. Al final del llistat, es posarà la quantitat d'hores i minuts totals, així com l'import total del treballador.

14. Horaris de l'institut

Volem crear una aplicació que ens permeti introduir les dades dels horaris dels grups de l'institut.

2ASIX						
DILLUNS		DIMECRES		DIJOUS		DIVENDRES
1DAM						
		DILLUNS	DIMARTS	DIMECRES	DIJOUS	DIVENDRES
8.00	8.00	LM Espe C22	PRG Abdó C22	ANG Ximo C22	LM Espe C22	FOL Antònia C22
8.55	8.55	FOL Antònia C22	PRG Abdó C22	SI Borja C22	LM Espe C22	EDD Fidel C22
9.50	9.50	FOL Antònia C22	BD Abdó C22	SI Borja C22	PRG Abdó C22	EDD Fidel C22
11.15	11.15	PRG Abdó C22	BD Abdó C22	EDD Fidel C22	PRG Abdó C22	ANG Ximo C22
12.10	12.10	PRG Abdó C22	SI Borja C22	PRG Abdó C22	BD Abdó C22	ANG Ximo C22
13.05	13.05	SI Borja C22	SI Borja C22	PRG Abdó C22	BD Abdó C22	BD Abdó C22

Per a cada grup volem guardar el codi del grup i el seu horari. Este horari ha de guardar informació de cada sessió de la setmana (5 dies x 6 franges horàries). No cal guardar en cap lloc els dies de la setmana ni les franges horàries.

Per a cada sessió caldrà guardar el codi del mòdul (assignatura), el nom del professor i el número de l'aula on s'impartirà la classe.

Tria l'estructura de dades més adient per a emmagatzemar tota eixa informació.

El programa tindrà un menú amb estes opcions:

1. Crear horari
2. Mostrar horari
3. Modificar horari

Caldrà definir i utilitzar adequadament estes següents:

- **getGrup**. Se li passa la llista de grups i un codi de grup. Retornarà l'objecte del grup corresponent. O null si no l'ha trobat.
- **creaGrup**. Se li passa la llista de grups. Es demanarà per teclat un codi de grup. Si ja està el grup en la llista (crida a getGrup), avisarà. Si no, per cada sessió de la setmana es preguntarà el mòdul, qui l'imparteix i a quina aula, i afegirà eixe grup a la llista. Retornarà l'objecte del grup creat (o null si no l'ha creat).
- **mostraHorari**. Se li passa la llista de grups, el codi de grup i quina dada d'estes 3 volem mostrar: (M)òdul/(P)rofe/(A)ula. Es mostrarà l'horari per pantalla (en forma de matriu) amb la informació corresponent. Si no existeix el grup retornarà false.
- **modificaHorari**. Se li passa la llista de grups i el codi del grup que volem modificar. Si existeix, es mostrarà el seu horari i es preguntarà pel dia de la setmana (1 a 5) i hora (1 a 6) que es vol modificar. A continuació es preguntarà l'assignatura, el professor i l'aula a canviar (si alguna d'estes dades es deixa en blanc, no es modificarà). A continuació s'actualitzaran les dades corresponents. Si no existeix el grup retornarà false.

A més d'eixes funcions, crea-te'n altres si creus que et fan falta.