

# Tema 8

## Estructures estàtiques. Registres. Classes i objectes

- 1 Introducció
- 2 Registres en Java. Classes i objectes
- 3 Operacions amb objectes
- 4 Registres niuats
- 5 Vectors de registres
- 6 Exercicis

### 1 Introducció

Per a guardar valors, en qualsevol llenguatge de programació, tenim:

- ✓ variables simples
- ✓ estructures estàtiques
  - taules o vectors (vist al tema 6)
  - registres



Una **taula** (vector o matriu) és una variable que pot guardar diverses dades però del mateix tipus: o tot enters, o tot cadenes, etc. Una taula és com una llista, on accedim a cada element per la posició que ocupa en la ella.

Un **registre** és una variable que pot guardar diverses dades encara que siguin de diferent tipus. Un registre és com una fitxa, on accedirem a cada element de la fitxa amb el nom del camp.



## 2 Registres en Java. Classes i objectes

Un registre és una variable composta per un conjunt de variables de **diferent tipus** que es poden referenciar sota el **mateix nom**.

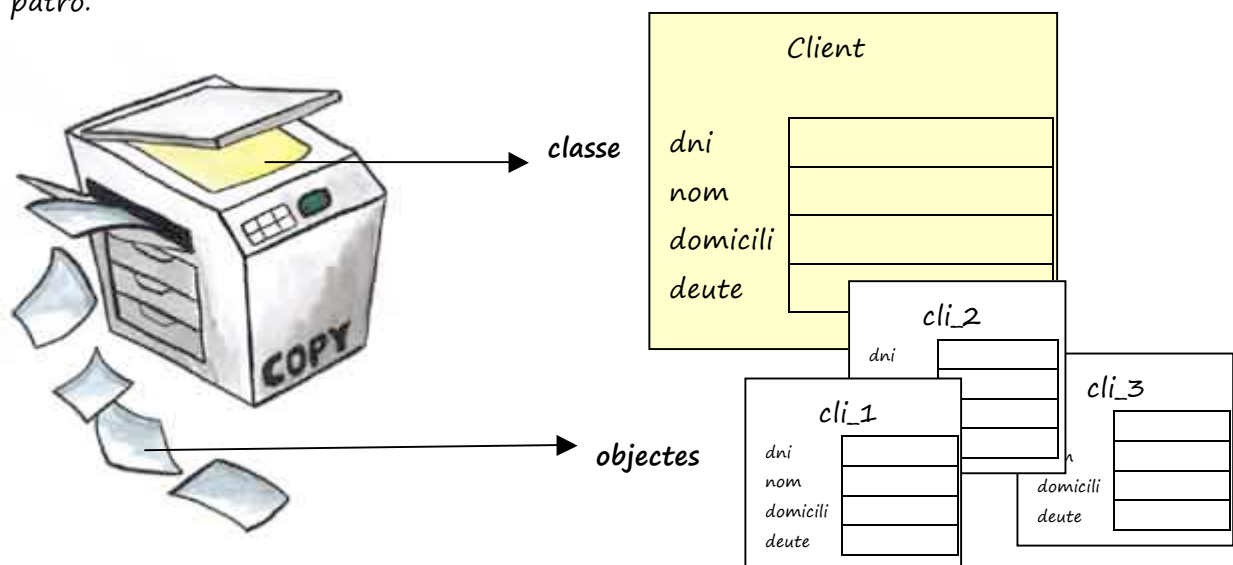
El motiu d'agrupar variables en un registre és que totes elles guarden informació relacionada (de la mateixa persona, objecte, etc).

Molts llenguatges poden implementar registres, encara que ho facen de forma diferent. Per exemple:

LLENGUATGE	TIPUS DE DADES PER ALS REGISTRES
Pascal	<i>record</i>
C	<i>struct</i>
Java	<i>class</i>

Java no té exactament un tipus de dades per als registres però ho implementa amb el que s'anomena *classes* i *objectes*. Una classe de Java és molt més potent que els registres de C i Pascal. Anirem afegint-li potència conforme avancem en el temari.

En Java, per a poder declarar una variable registre (objecte), primer necessitem crear un motle o patró (classe), per a després definir els nostres registres en base a eixe patró.



Imaginem que la *classe* *Client* és una fitxa plastificada (sense omplir), és a dir, una plantilla. Com que està plastificada, no podem escriure damunt, però podem fer totes les fotocòpies que vulguem (*objectes*) per a posar les dades de cada client.

És a dir, podem establir una correspondència entre els següents termes:

ANALOGIA EN LA VIDA REAL	TERMINOLOGIA INFORMAL	TERMINOLOGIA FORMAL	EXEMPLES
Fitxa plastificada Plantilla Motle Patró	Tipus de registre	<b>Classe</b>	Client
Apartats de la fitxa Característiques Camps	Variables membre	<b>Atributs</b>	dni, nom, domicili, deute
Fotocòpies	Variables de registre	<b>Objectes</b>	cli_1, cli_2, cli_3

Anem a veure com podem definir en Java les classes i els objectes. Ara bé, cal tindre en compte que una classe és més que un conjunt d'atributs, però això ja ho vorem més endavant.

## 2.1 Definició de la classe

Per a crear eixe motle (classe) es fa amb la paraula *class*. I dins definirem els camps (atributs o variables membre) que inclou.

Sintaxi:

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

Exemple:

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

La classe *client* és un motle per a poder crear variables de registres (objectes) on guardar les dades de cada client.

## Exercicis

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

## 2.2 Definició dels objectes

Abans hem vist com crear en Java la *classe client*. És a dir, ja hem creat la plantilla o estructura per a crear fitxes de clients. Ara necessitaré crear variables de registre per a gestionar els clients. És a dir: anem a fer fotocòpies de la plantilla. Crearem objectes a partir de la classe clients.

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	1r) Definim l'objecte i li reservem memòria en una sola instrucció.
Sintaxi	NomClasse <u>nomObjecte</u> ; <u>nomObjecte</u> = new NomClasse();	NomClasse <u>nomObjecte</u> = new NomClasse();
Exemple	Cotxe <u>meu</u> , <u>teu</u> ; <u>meu</u> = new Cotxe(); <u>teu</u> = new Cotxe();	Cotxe <u>meu</u> = new Cotxe(); Cotxe <u>teu</u> = new Cotxe();

## Exercicis

2. Modifica el projecte *Proves*: crea, dins del main, un objecte per a cadascuna de les classes que ja has definit, de la següent forma:

- ✓ Crea els objectes de les primeres 4 classes amb instruccions separades per a definició de l'objecte i reserva de memòria.
- ✓ Crea els objectes de les altres classes amb una sola instrucció per a definició de l'objecte i reserva de memòria.

## 2.3 Utilització dels objectes

Ja hem creat la classe (el nou tipus) i els objectes (variables d'eixe nou tipus). Ara anem a utilitzar eixos objectes. És a dir, anem a vore com posar valors en els seus atributs (variables membre), mostrar-los per pantalla, consultar-los, etc.

Exemples d'ús:

```
System.out.println("DADES DEL CLIENT:");
System.out.println("DNI: ");      cli_1.dni = llegirEnter();
System.out.println("Nom: ");      cli_1.nom = llegirCadena();
System.out.println("Adreça: ");   cli_1.domicili = llegirCadena();
System.out.println("Deute: ");    cli_1.deute = llegirReal();
...
cli_2.deute = cli_2.deute - cli_1.deute;
if (cli_2.deute > 6000) {
    System.out.println("El client " + cli_2.nom + " deu " + cli_2.deute + " euros.");
}
```

De moment hem vist com usar els atributs dels objectes (igual que s'usen les variables normals). Més avant vorem com usar els propis objectes.

Anem a vore un exemple complet de definició de classes, objectes i ús:

```

import java.io.*;

//***** CLASSE Alumne *****/
// ----- Definició de la classe que fa de registre -----
class Alumne {
    String nom;
    String cognom;
    String [] telefon = new String [3]; // un vector de 3 possibles telèfons
    int edat;
}

//***** 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();
        // ----- Ús dels objectes (més bé, ús dels seus atributs) --
        a1.nom = "Miquel Josep";
        a1.cognom = "Garcia";
        a1.edat = 5;
        a1.telefon[0] = "961712222";
        a1.telefon[1] = "961702299";
        System.out.println(a1.nom);
        System.out.println(a1.cognom);
        System.out.println(a1.edat);
        for (int i=0; i<a1.telefon.length; i++) {
            System.out.println(a1.telefon[i]);
        }
    }
}

```

## Exercicis

3. Modifica el projecte *Proves* per a assignar valors als objectes i mostrar el seu resultat per pantalla.

### 3 Operacions amb registres (objectes)

Com s'utilitzen els objectes? De 2 formes:

- Individualment, amb cada atribut de l'objecte (ja vist)
- Fent referència a tot l'objecte en conjunt

#### 3.1 Operacions amb els atributs dels objectes

Com ja hem vist anteriorment, les operacions que podem fer amb els atributs dels objectes són les mateixes que fem amb una variable simple normal, però amb el nom de l'objecte, un punt i el nom de l'atribut.

Exemples:

- ✓ `a1.nom = "Pep";` // assignació directa
- ✓ `a1.cognom = llegirCadena();` // assignació per teclat
- ✓ `System.out.println(a1.telefon[1]);` // impressió
- ✓ `a1.telefon[1] = a1.telefon[0];` // còpia
- ✓ `a1.edat++;` // autoincrement
- ✓ `if (a1.edat > 17) ...` // consulta

Però ara vorem com treballar no amb els atributs dels objectes sinó amb els objectes en conjunt.

## 3.2 Operacions amb l'objecte en conjunt

Abans de vore les operacions amb objectes en conjunt, hem de tindre en compte que UN OBJECTE ÉS UNA ADREÇA DE MEMÒRIA on estan les seues dades.

Vegem-ho amb un exemple:

// Definició de la classe Alumne

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

// Definició de 2 objectes (2 alumnes)

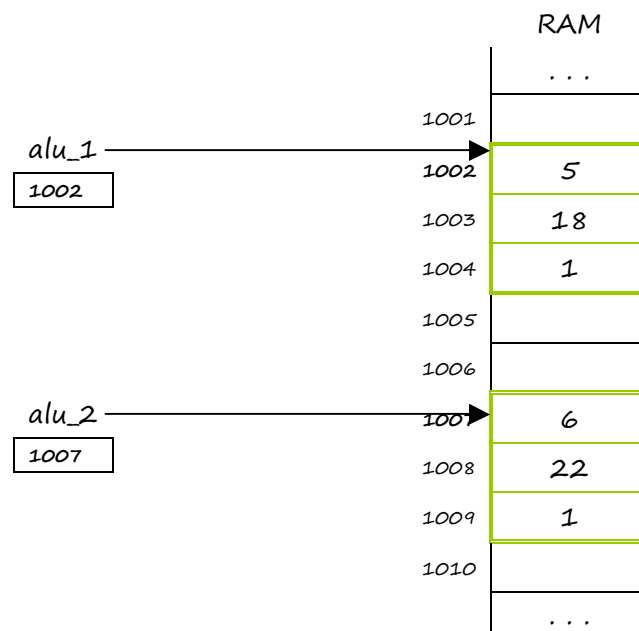
Alumne alu\_1 = new Alumne(); // Imaginem que se li assigna l'adreça 1002

Alumne alu\_2 = new Alumne(); // Imaginem que se li assigna l'adreça 1007

// Inicialització dels 2 objectes

alu\_1.num = 5;      alu\_1.edat = 18;      alu\_1.curs = 1;

alu\_2.num = 6;      alu\_2.edat = 22;      alu\_2.curs = 1;



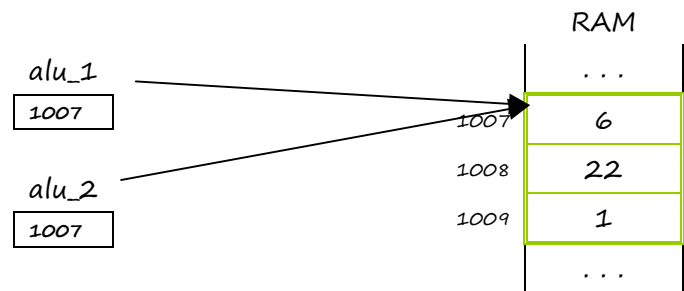
Per tant, hem d'anar amb compte amb les assignacions i comparacions entre objectes, ja que, possiblement, no estarem fent allò que pretenem. Ara ho vorem.



## Copiar objectes

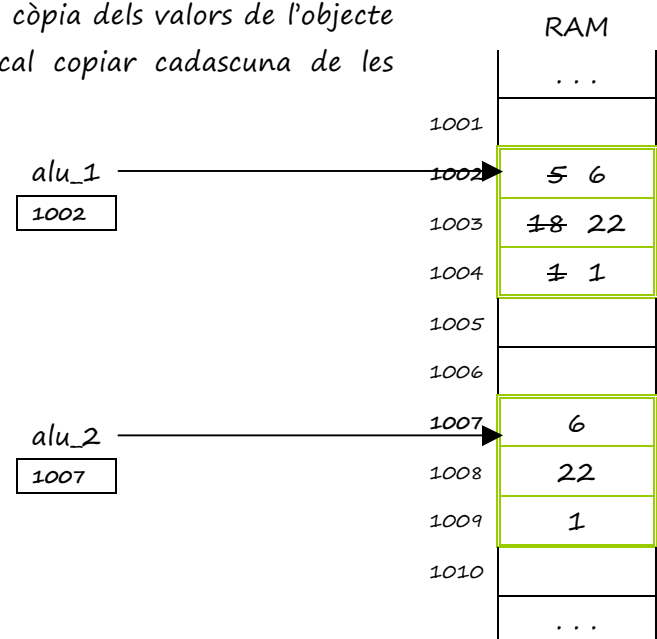
Què passa si copiem objectes amb `alu_1 = alu_2;` ?

Si després modifiquem algun component d'un dels dos objectes, l'altre també tindrà eixes modificacions, ja que els dos objectes estan apuntat a la mateixa zona de memòria on estan els valors.



Solució : si el que volíem fer és una còpia dels valors de l'objecte (i no que apunten al mateix objecte), cal copiar cadascuna de les variables membre:

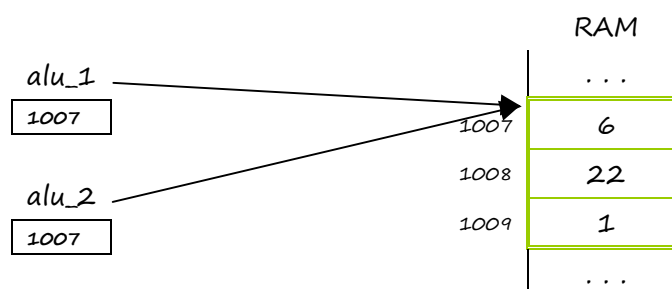
```
alu1.num = alu2.num;  
alu1.edat = alu2.edat;  
alu1.curs = alu2.curs;
```



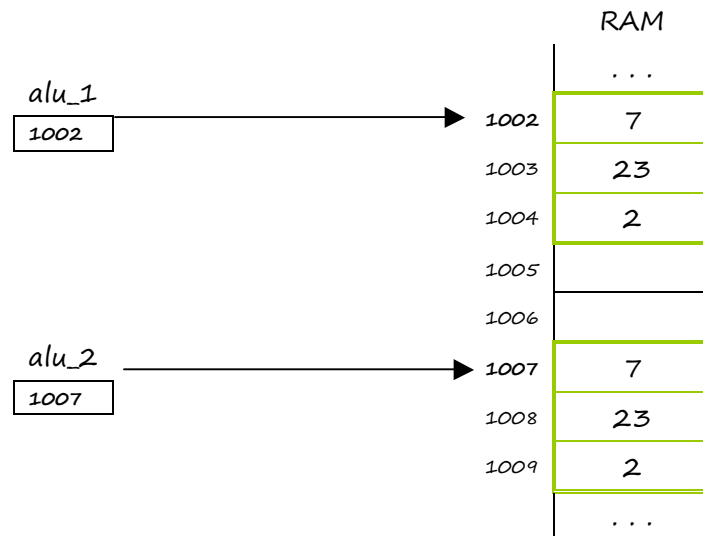
## Comparar objectes

Què passa si comparem objectes amb `(alu_1 == alu_2)` ?

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



b) Però si apunten a zones diferents, serà *false*, encara que els valors dels components dels objectes siguin els mateixos:



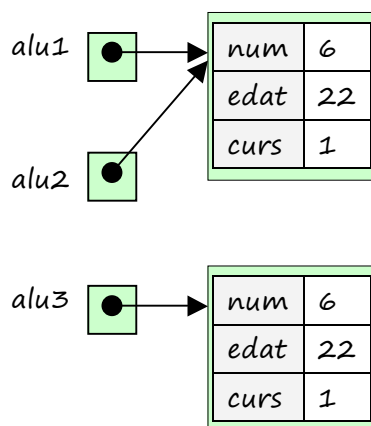
Solució : si el que volíem fer és comparar els valors dels 2 objectes (i no comparar que apunten al mateix objecte), cal comparar cadascuna de les variables membre:

¿ (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 de les seues variables membre 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:

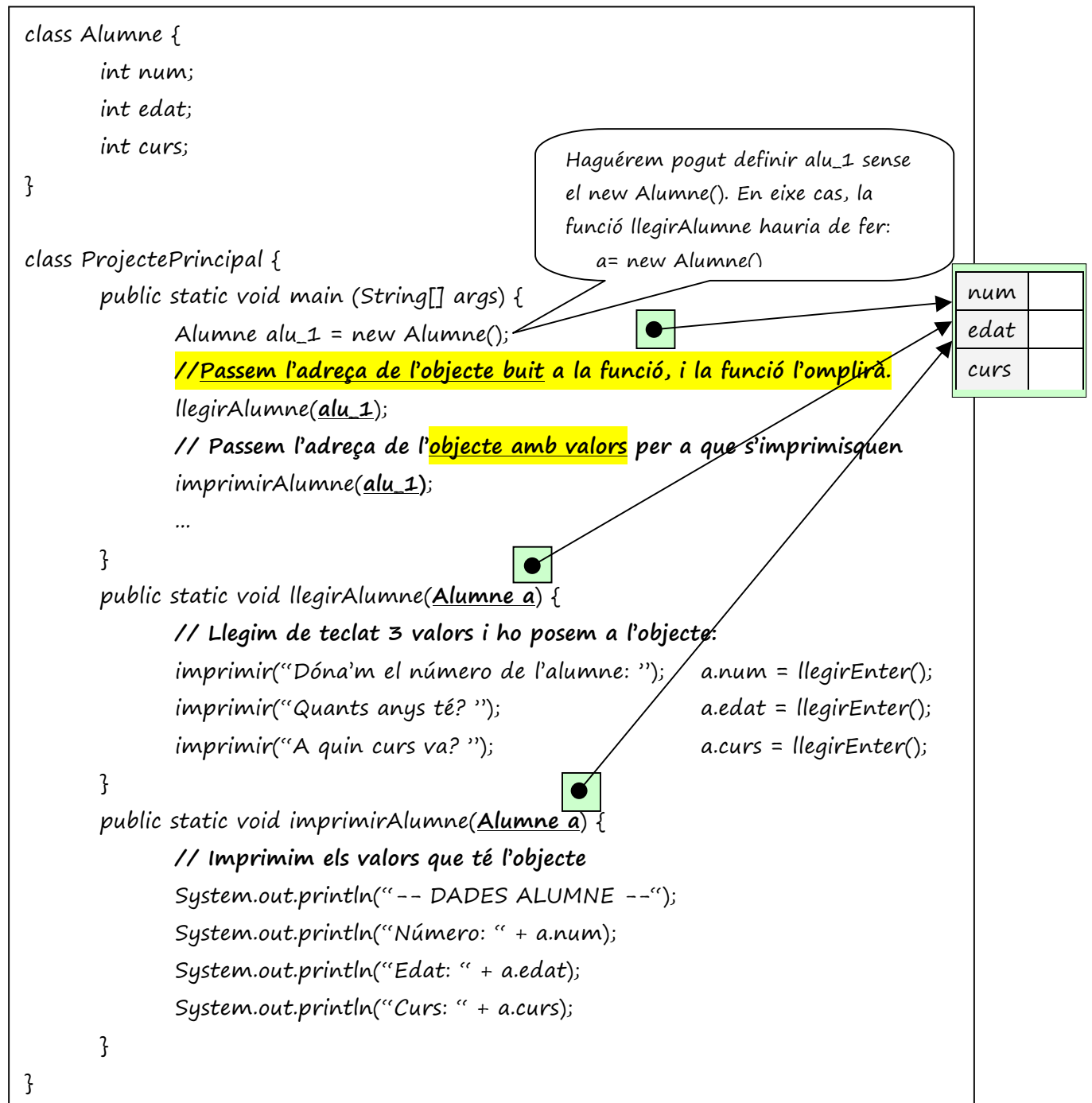


Segons eixa situació, observem que:

- ✓ alu1 == alu2
- ✓ alu1 != alu3
- ✓ alu2 != alu3
- ✓ Si modifique *alu2.edat*, TAMBÉ estic modificant *alu1.edat*
- ✓ Si modifique *alu2.edat*, NO estic modificant *alu3.edat*

## 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.



## Exercicis

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.

## 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) {
```

No reservem memòria per a alu\_1 ja que s'encarregarà la funció `llegirAlumne()`.

**// Anem a demanar dades d'un alumne nou (i guardar-lo en alu\_1):**

```
Alumne alu_1;  →  
alu_1 = llegirAlumne();
```

**// Anem a copiar alu\_1 a alu\_2 (però en diferents zones de memòria):**

```
Alumne alu_2;  →  
alu_2 = clonarAlumne(alu_1);  
...
```

No reservem memòria per a alu\_2 ja que s'encarregarà la funció `clonarAlumne()`.

```
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? ");
```

```
    return a;
```

```
}
```

```
public static Alumne clonarAlumne(Alumne a) {
```

```
    Alumne aCopiat = new Alumne(); 
```

```
    aCopiat.num = a.num;
```

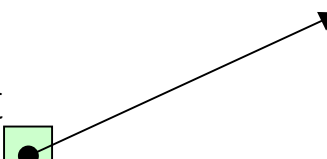
```
    aCopiat.edat = a.edat;
```

```
    aCopiat.curs = a.curs;
```

```
    return aCopiat;
```

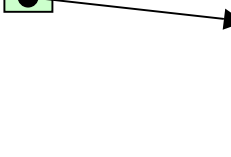
```
}
```

```
}
```



num	
edat	
curs	

```
a.num = llegirEnter();  
a.edat = llegirEnter();  
a.curs = llegirEnter();
```



num	
edat	
curs	

## Exercicis

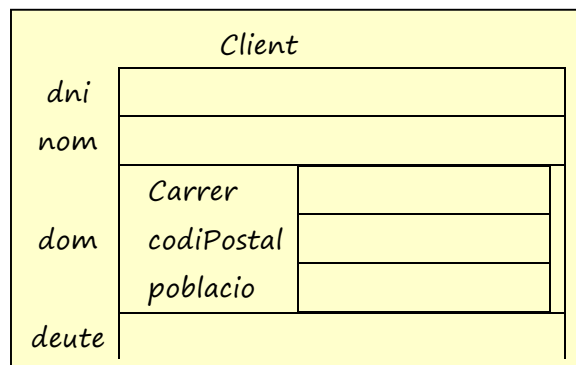
Modifica el projecte proves:

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:
    - i. Crear una altra data, de nom `d_nova` (declarar-la i reservar-li memòria amb `new`).
    - ii. Copiar en eixa altra data (`d_nova`) les dades de la data del paràmetre però amb un any més.
    - iii. Retornar eixa nova data.
  - b. En el main crea una data (dis-li de nom: `d_origen`), posa-li valors i crida a la funció `incrementaAnyEnDataNova` passant-li eixa data com a paràmetre. Recorda que hauràs d'arreplegar la data retornada. Per a això, caldrà crear-te prèviament una altra data (dis-li de nom: `d_destí`), la qual no caldrà que li reserves memòria amb el `new`.
  - c. Comprova en el main que la nova data (`d_destí`) té els valors esperats: els mateixos que `d_origen` però amb un any més (mostra-ho per pantalla).

## 4 Registres niuats

Ja hem vist que un registre és una estructura que conté a altres variables, les quals poden ser de qualsevol tipus. Per tant, un d'eixos tipus també pot ser un registre. És a dir, tindrà un registre dins d'un altre. Això s'anomena un *registre niuat*.

Exemple:



Implementació:

### 1) Crear classe Domicili

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

### 2) Crear classe Client

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

O bé:

```
Domicili dom = new Domicili();
```

Ara vorem la diferència.

a) Definir en la classe *Client* un objecte *Domicili* sense instanciar (sense *new*):

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

Ací NO posem:  
= new Domicili()

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

```
        Client cli1 = new Client();
```

Ací es reserva memòria  
per al client però no per  
al seu domicili.

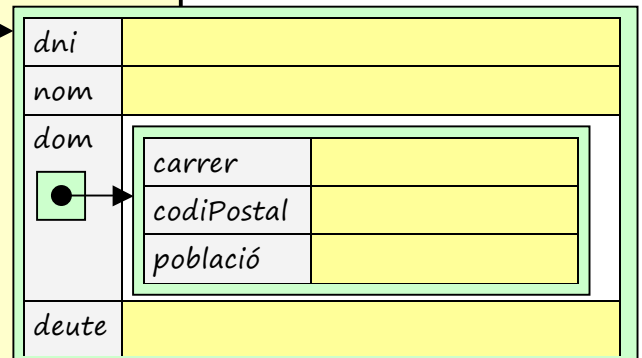
cli1



```
        cli1.dom = new Domicili();
```

Si volem posar dades en el  
domicili del client, ací es on  
haurem de reservar memòria  
per al domicili d'eixe client.

cli1



```
        cli1.dom.codPostal = 46410;
```

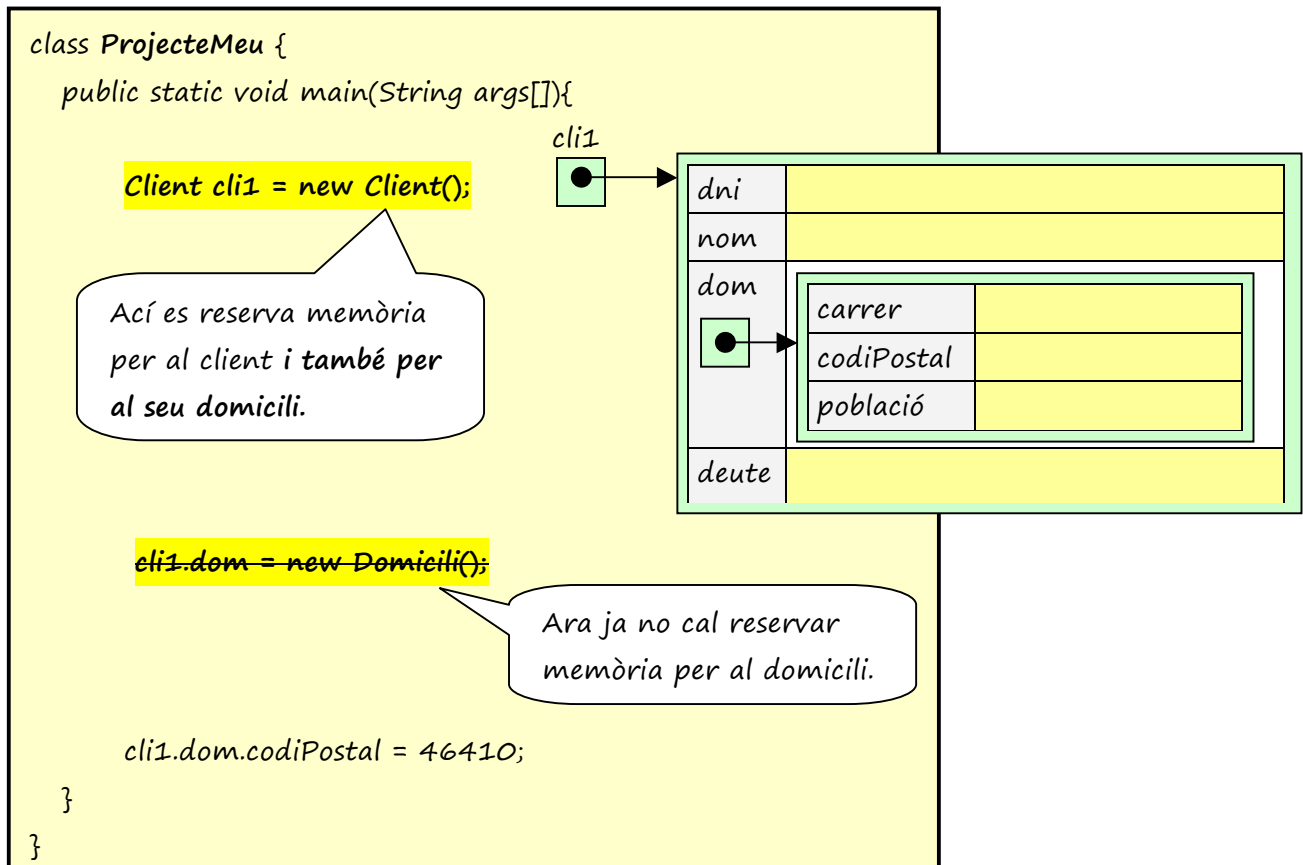
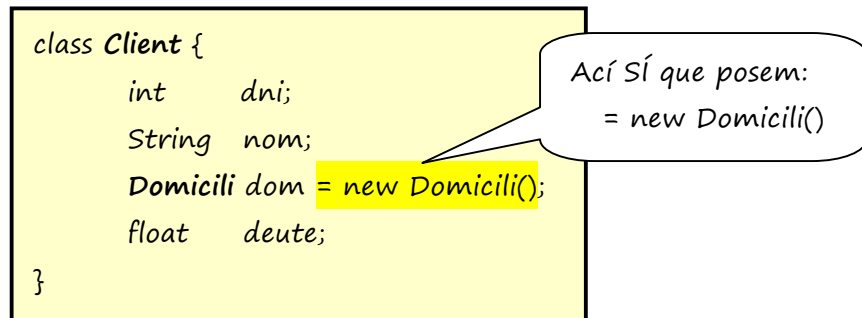
```
    }
```

```
}
```



Espai que es resreva amb el *new*.

b) Definir en la classe *Client* un objecte *Domicili* instanciant (amb new):



Quan definir un registre niuat amb el new i quan sense new?

- **Sense new.** Si anem a tindre molts objectes de la classe *Client* i cap la possibilitat que pocs d'ells tinguin domicili, seria una reserva de memòria innecessària per a tots els domicilis (estarien "buits"). En eixe cas només reservariem memòria per al domicili quan fera falta.

- **Amb new.** Si no és el cas anterior, farem que quan es reserve memòria per a un client, que automàticament també es reserve memòria per al seu domicili, ja que ens serà més fàcil de programar.

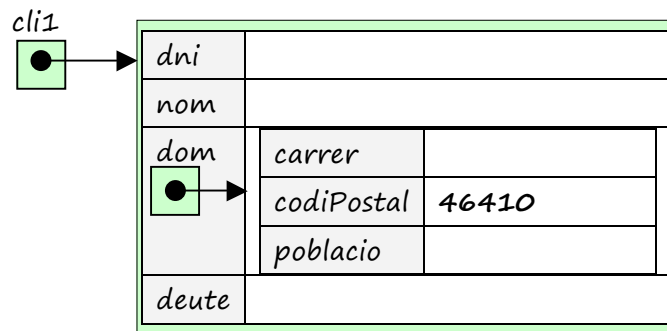


Nota: veiem que, quan usem el “punt” ( . ) estem fent ús de les “fletxes” ( → ):

`cli1.dom.codiPostal = 46410;`

és com fer:

`cli1→dom→codiPostal = 46410;`



## Exercicis

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).

NomComplet

nom	<input type="text"/>
cog1	<input type="text"/>
cog2	<input type="text"/>

Domicili

carrer	<input type="text"/>
numero	<input type="text"/>
pis	<input type="text"/>
porta	<input type="text"/>
---	
comarca	<input type="text"/>

TelFix

prefix	<input type="text"/>
numero	<input type="text"/>

- 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.

(Alumne)

nomC	(NomComplet)
	nom
	cog1
dom	(Domicili)
	carrer
	numero
	---
telPar	(TelFix)
	prefix
telMar	(TelFix)
	prefix

- 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.

(Ordinador)

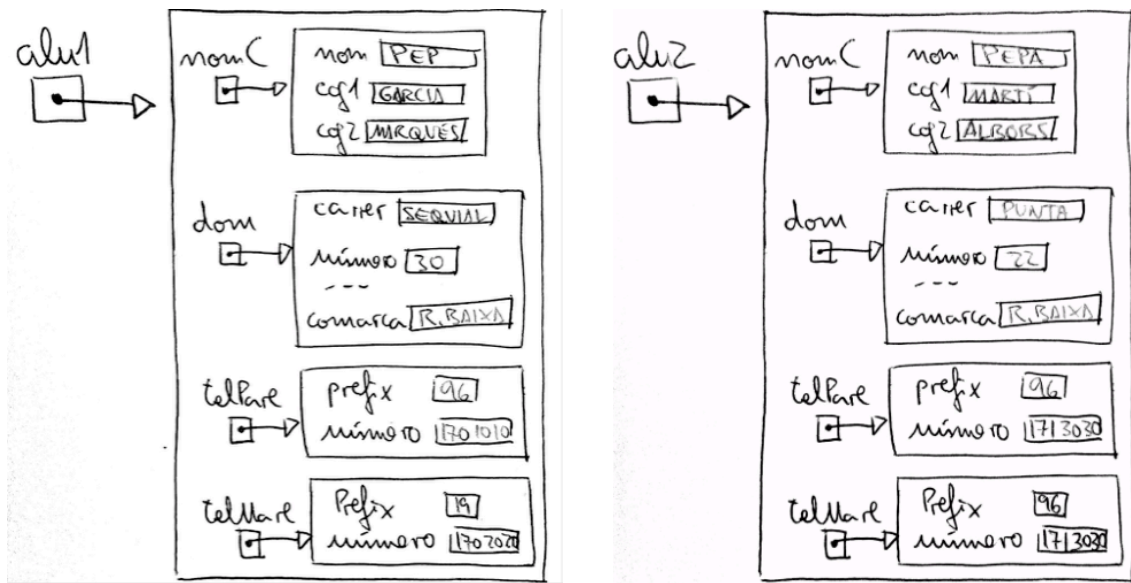
nSerie
cpu
ram
dd
alu

(Alumne)

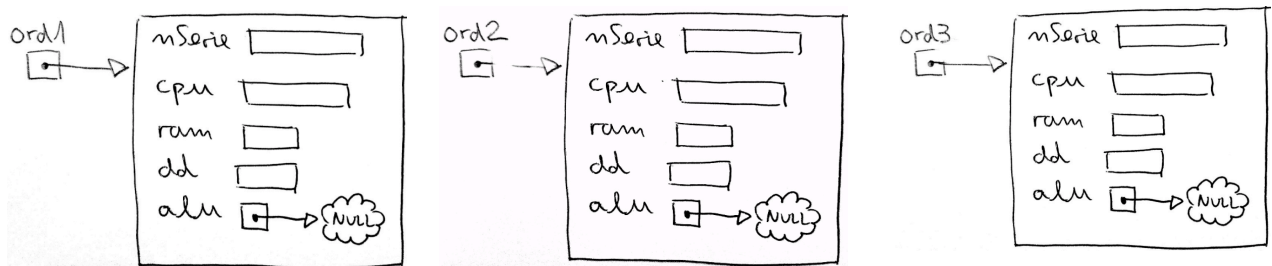
nomC	(NomComplet)
	nom
	cog1
dom	(Domicili)
	carrer
	numero
	---
telPar	(TelFix)
	prefix
telMar	(TelFix)
	prefix

9. En el projecte Institut, en el main:

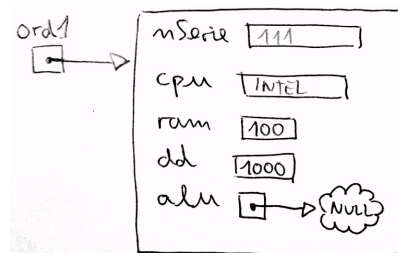
a. Crea 2 alumnes (alu1 i alu2) i ompli'ls de dades qualssevol.



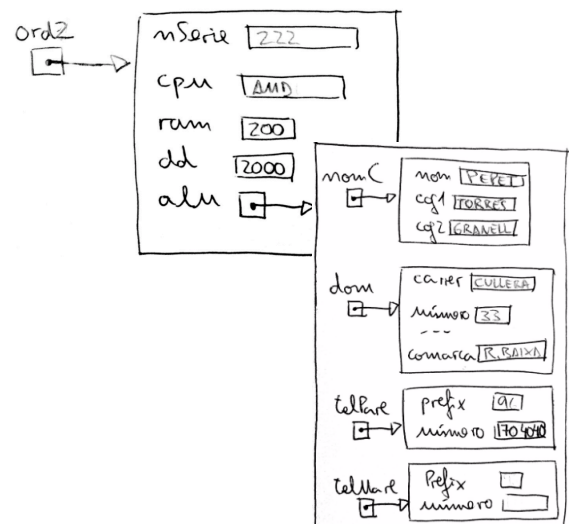
b. Crea 3 ordinadors (ord1, ord2 i ord3), sense omplir-los de dades.



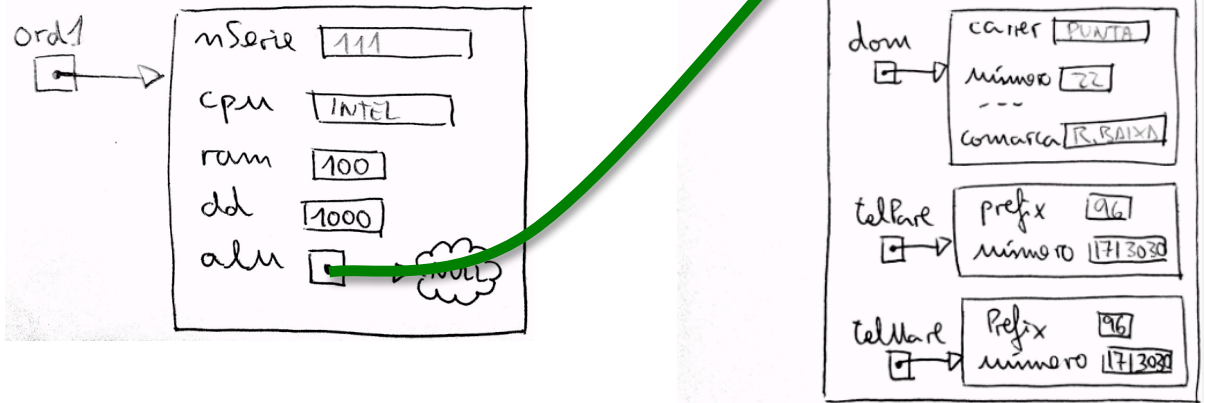
c. Ompli ord1 de dades qualssevol però sense assignar-li cap alumne.



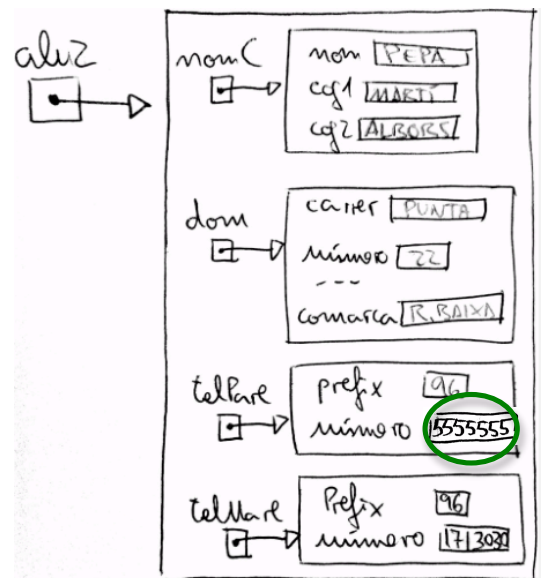
d. Ompli ord2 de dades qualssevol assignant-li les dades d'un nou alumne.



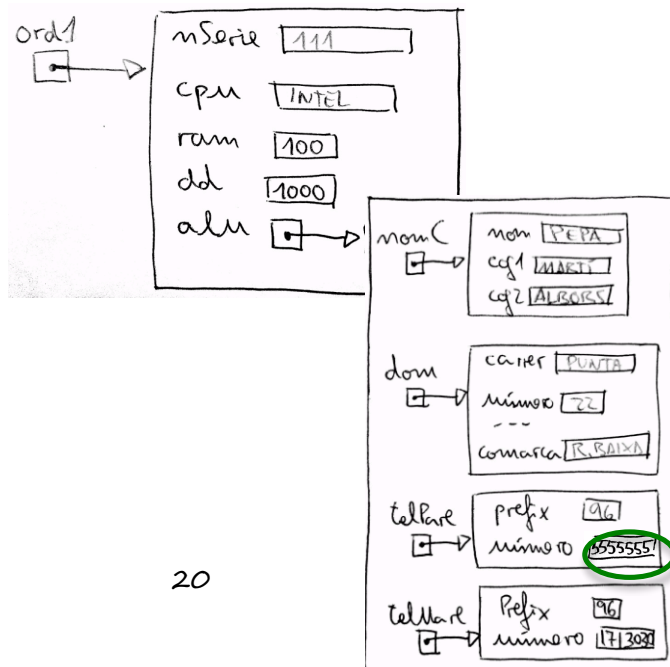
e. Assigna a l'ordinador 1 l'alumne 2.



f. Canvia el número de telèfon del pare de l'alumne 2 (per exemple: 555555)



g. Mostra per pantalla el telèfon del pare de l'alumne que està usant l'ordinador 1. No ho faves a partir de l'alumne 2, sinó a partir de l'ordinador 1. Comprova que ha canviat (ha de mostrar 555555).



## 5 Vectors de registres

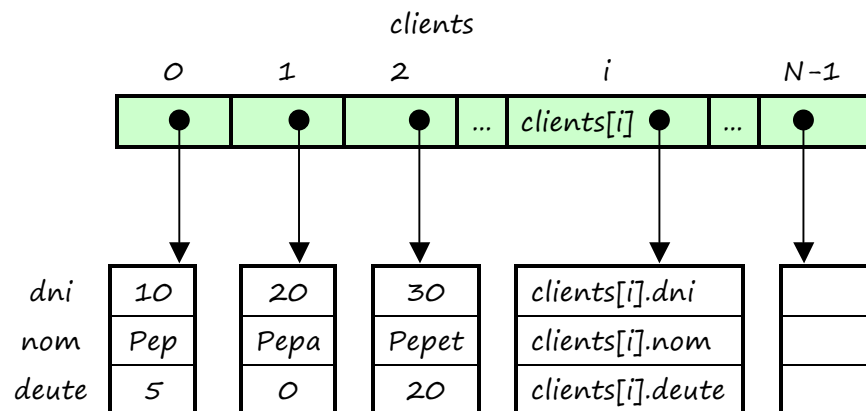
### Definició

L'ús simultani de vectors i registres proporciona una poderosa eina per a emmagatzemar i manipular informació.

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

	clients				
	0	1	2	...	$N-1$
dni	10	20	30		
nom	Pep	Pepa	Pepet		
deute	5	0	20		

Com ja hem dit anteriorment, hem de tindre clar que, per exemple, `clients[2]` no és l'objecte que està a la tercera posició del vector, sinó una referència (adreça) d'on està eixe objecte. És a dir:



### Implementació

Per a crear eixa estructura haurem de:

- 1r) Definir la classe *Client* (el tipus registre)
- 2n) Definir un vector de registres de tipus *Client*
- 3r) Reservar memòria per a cadascun dels registres del vector

Vegem amb un exemple com es crea i utilitza un vector de registres.

Exemple: vector de 100 clients

// 1r) Definim la classe Client

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

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

// 2n) Definim un vector de N clients

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

// 3r) Reservem memòria per a cadascun dels N objectes

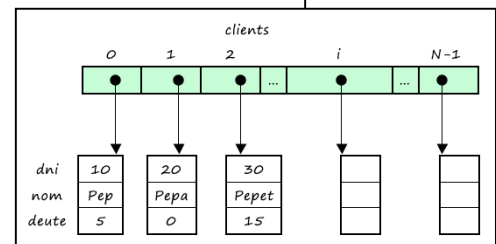
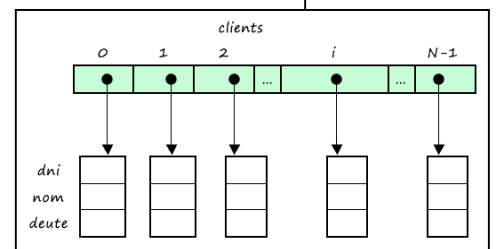
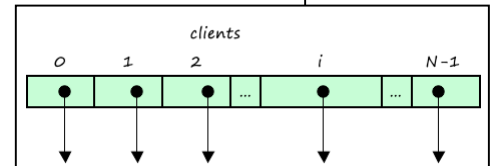
```
for (int i=0; i<N; i++)  
    clients[i] = new Client();
```

// Omplim el vector de clients amb dades llegides de teclat:

```
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 que acabem de posar al vector de 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 de registres, o registres que tenen alguna matriu com a variable membre, etc.

## Exercicis

10. Una biblioteca vol emmagatzemar informació de cada llibre que té: codi de referència (alfanumèric), autors (màxim, 3), títol, editorial i any. Implementa l'estructura necessària per a guardar 100 llibres. Reserva memòria per a cadascun d'eixos llibres. Després dóna d'alta un llibre qualsevol.
11. Un empresa de lloguer de cotxes vol tindre guardada la informació de cadascun dels cotxes: matrícula (lletres i número), marca, model, data de compra i kms. Implementa l'estructura necessària per a guardar 100 cotxes (però no reserves memòria per a tots els cotxes). Després dóna d'alta un cotxe qualsevol. Comprova que no tens errors de compilació ni d'execució.
12. L'empresa constructora "MACA S.A." promou la construcció de l'edifici de luxe "Xequin-Pis". L'edifici està organitzat en 6 escales (de la A a la 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.



- Definix la classe **Vivenda** amb les variables: *m2*, *q\_hab*, *preu*, *nom*, *nif*
- Definix **edifici** (al *main*), com una matriu tridimensional de vivendes. Utilitza les constants necessàries per a establir les dimensions de la matriu.
- 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.
- 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. 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

1. Demanar dades d'una feina i afegir-la en un vector de feines.

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. Volem crear una aplicació que ens permeti introduir les dades dels horaris dels grups de l'institut (màxim 100).

Per a cada grup volem guardar el codi i l'horari. I este horari estarà guardat en forma de matriu (5 dies x 6 franges horàries).

En cada cel·la de la matriu caldrà guardar el codi de l'assignatura, el nom del professor i el número de l'aula on es donarà la classe:

Nota: no cal guardar en cap lloc els dies de la setmana ni les franges horàries.

Hores	DILLUNS	DIMARTS	DIMECRES	DIJOUS	DIVENDRES
8:00 - 8:55	SIMM Alicia 42	SIMM Alicia 42	PRO Abdó 42	XAL Juanjo 42	XAL Juanjo 42
8:55 - 9:50	SIMM Alicia 42	SIMM Alicia 42	PRO Abdó 42	XAL Juanjo 42	XAL Juanjo 42
9:50 - 10:45	XAL Maite 42	PRO Abdó 42	SIMM Alicia 42	RET Bataller 42	PRO Abdó 42
11:15 - 12:10	XAL Maite 42	PRO Abdó 42	SIMM Alicia 42	RET Bataller 42	PRO Abdó 42
12:10 - 13:05	FOL Maravilla 42	PRO Abdó 42	XAL Maite 42	PRO Abdó 42	SIMM Alicia 42
13:05 - 14:00	FOL Maravilla 42	XAL Maite 42	XAL Maite 42	PRO Abdó 42	SIMM Alicia 42

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

El programa tindrà el menú següent:

- 1) Crear horari
- 2) Mostrar horari
- 3) Modificar horari

Caldrà definir i utilitzar adequadament les funcions següents:

- ✓ **crearHorari:** es demanarà per teclat un codi de grup i el seu horari i es guardarà a la llista de grups (que se li passa com a paràmetre a la funció). Si no cabera retornarà false.
- ✓ **mostrarHorari:** es passa la llista de grups, el codi de grup i quina dada de les 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.
- ✓ **modificarHorari:** es 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.