

# Tema 10

## Estructures estàtiques. Registres

- 1 Introducció
- 2 Registres en Java
- 3 Operacions amb registres (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.

TAULA ≈ LLISTA



Formulari de registre de treballador. Dades: GONZALEZ LOPEZ ADELA, N.º Cartilla Seg. Enferm. 1.054.922, N.º Contrato de trabajo A. 171, Esparraguera Barcelona 32, Luis y Adela, Esparraguera Barcelona 12-10-1936, 19-10-1957, Motivo Voluntaria, Ingreso en el trabajo 17-2-1953, Baja 19-10-1957, Reingreso, Telares, aprendizaje-tejedora, Ver dorso, ¿Trabajó anteriormente en esta fábrica? - ¿Cuándo? - (Trabajo últimamente en) (Hasta cuando) Otros datos de interés:

Un **registre** és una variable que pot guardar diverses dades encara que siguin de diferent tipus. S'utilitza per a encapsular informació relacionada.

REGISTRE ≈ FITXA

## 2 Registres en Java

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

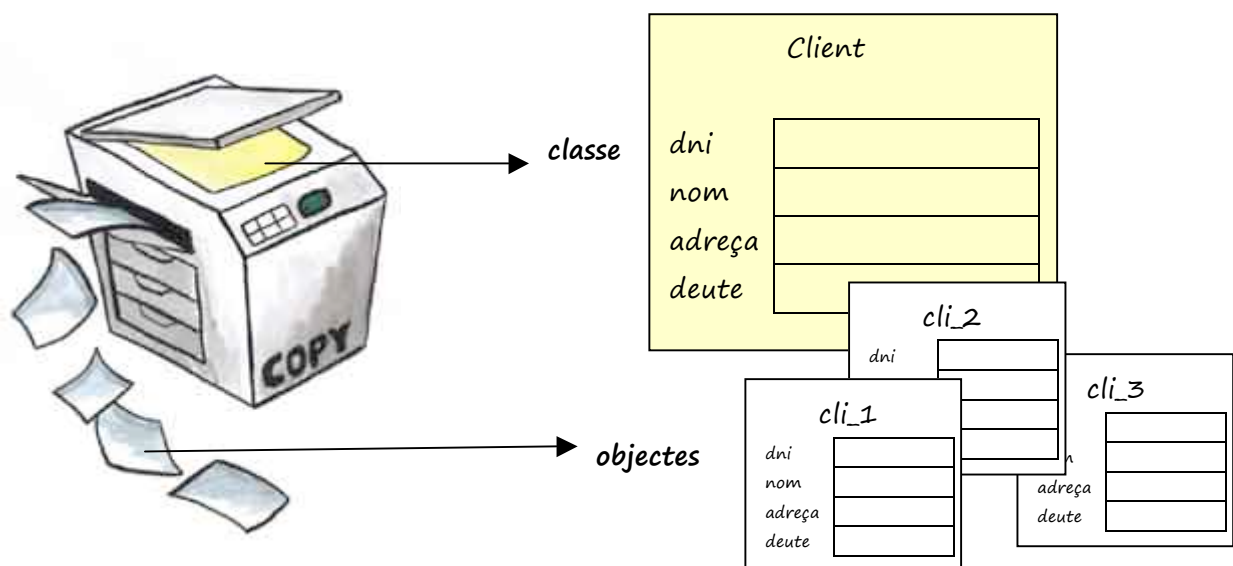
El motiu d'agrupar variables en un registre és que totes elles guarden informació relacionada (de la mateixa persona, objecte, etc).

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

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

Java no té un tipus de dades per als registres però ho pot implementar 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, adreça, 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 adreça;  
    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.
<u>Sintaxi</u>	NomClasse <u>nomObjecte</u> ; <u>nomObjecte</u> = new NomClasse();	NomClasse <u>nomObjecte</u> = new NomClasse();
<u>Exemple</u>	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.adreça = 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)

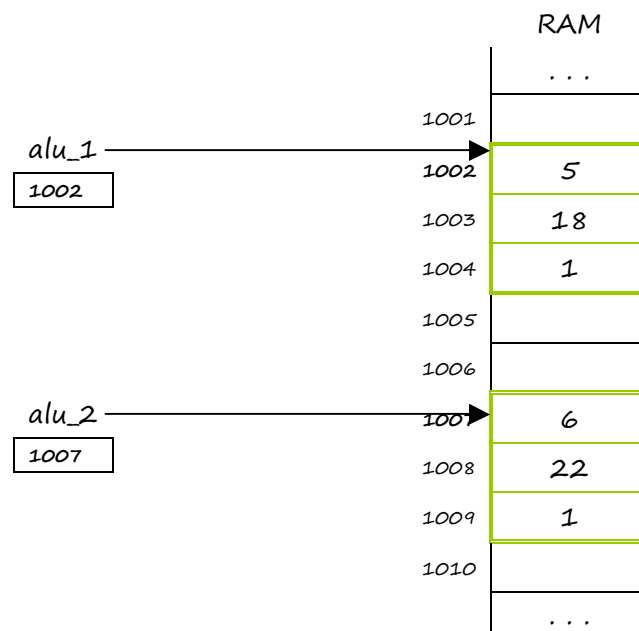
Alumne alu\_1 = new Alumne(); // Imaginem que se li assigna l'adreça 1002

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;



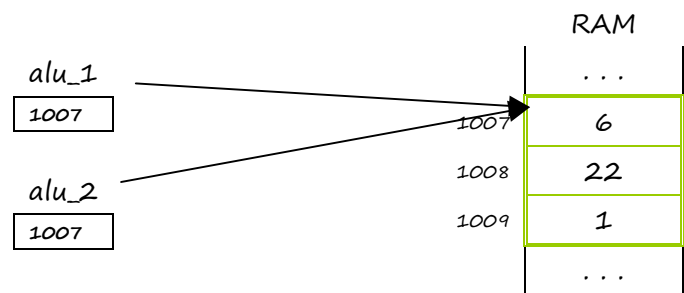
Per tant, hem d'anar amb compte amb les assignacions i comparacions entre objectes, ja que, possiblement, no estarem fent allò que pretenem. Ara ho vorem.



## Copiar objectes

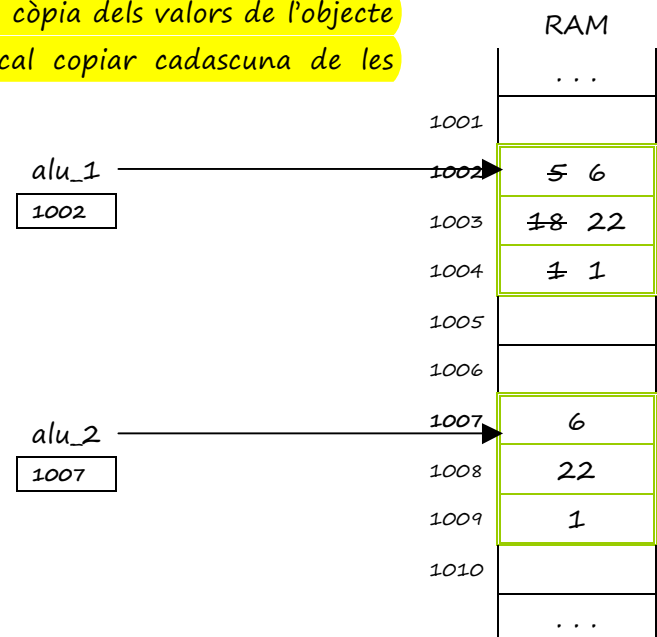
Què passa si copiem objectes amb `alu_1 = alu_2;` ?

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:

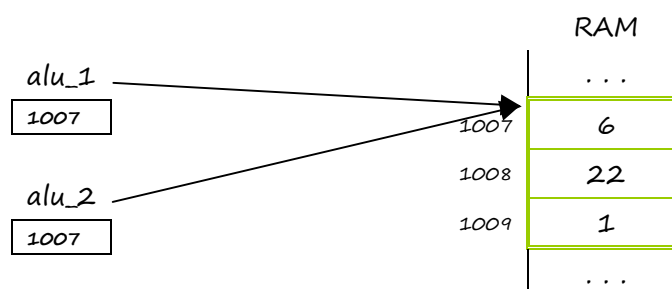
```
alu1.num = alu2.num;  
alu1.edat = alu2.edat;  
alu1.curs = alu2.curs;
```



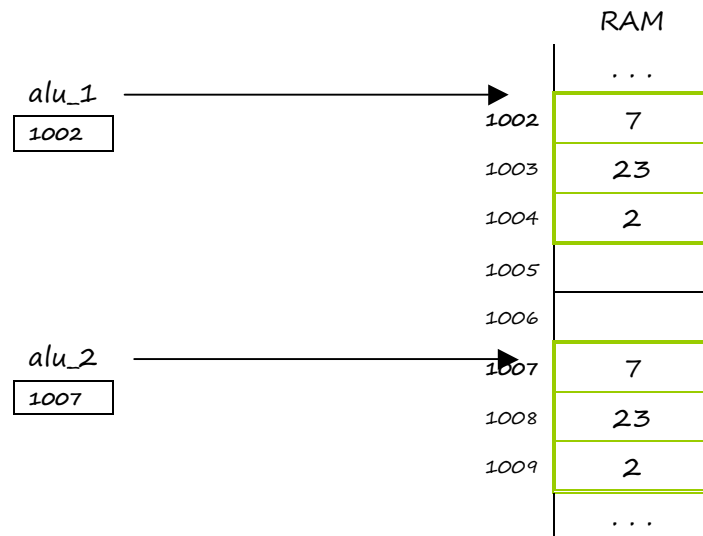
## Comparar objectes

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:



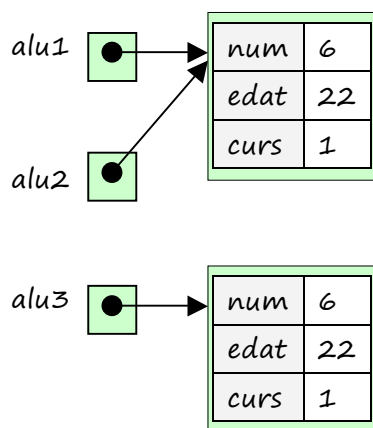
**Solució :** si el que volíem fer és comparar els valors dels 2 objectes (i no comparar que apunten al mateix objecte), cal comparar cadascuna de les variables membre:

¿ (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.

```
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

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 ho farà la funció:  
        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);  
        ...  
    }  
  
    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;  
    }  
}
```

Tampoc reservem memòria per a alu\_2 ja que ho farà la funció.

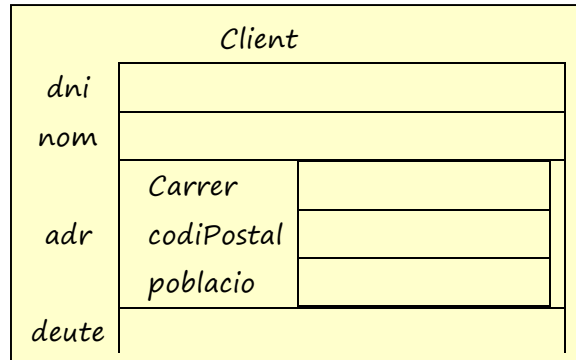
num	
edat	
curs	

num	
edat	
curs	

## 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 Adreça

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

2) Crear classe Client, basant-nos en Adreça. Es pot fer de 2 formes:

a) Reservant ja memòria per a l'adreça. No la reservarem en els objectes:

```
class Client {  
    int    dni;  
    String nom;  
    Adreça adr = new Adreça();  
    float  deute;  
}
```

// Dins de la funció main:

```
// Reserva de memòria per a un client i adreça:  
Client cli1 = new Client();  
// Ací JA NO reservem memòria per a l'adreça  
// introduir dades  
cli1.adr.codiPostal = 46410;
```

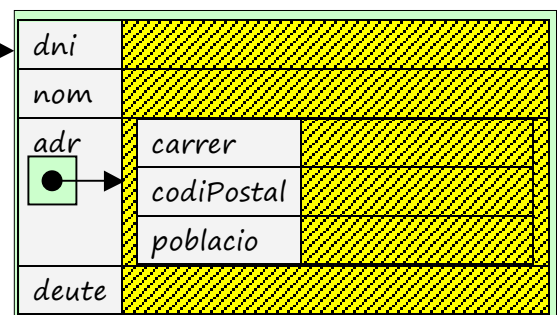
cli1



Client cli1 = new Client();



Zona que es reserva amb el new Client();



b) Sense reservar memòria per a l'adreça. La reservarem en els objectes:

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

// Dins de la funció main:

// Reserva de memòria per a un client:

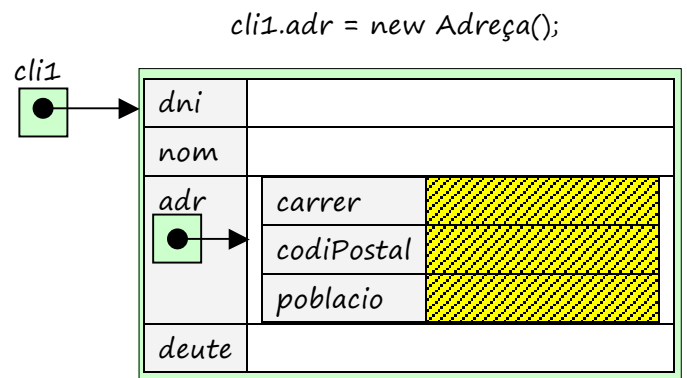
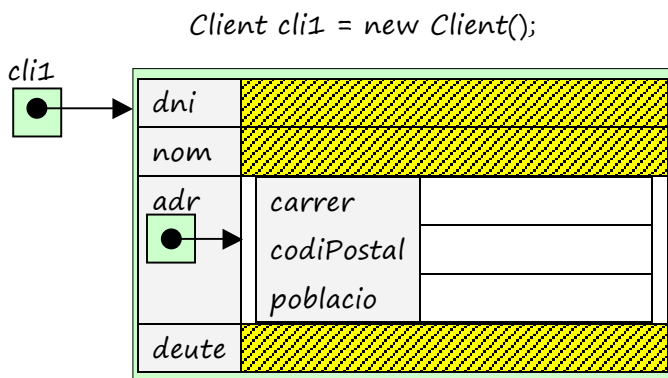
**Client cli1 = new Client();**


// Reserva de memòria per a l'adreça d'eixe client:

**cli1.adr = new Adreça();**

// introduir dades

cli1.adr.codiPostal = 46410;



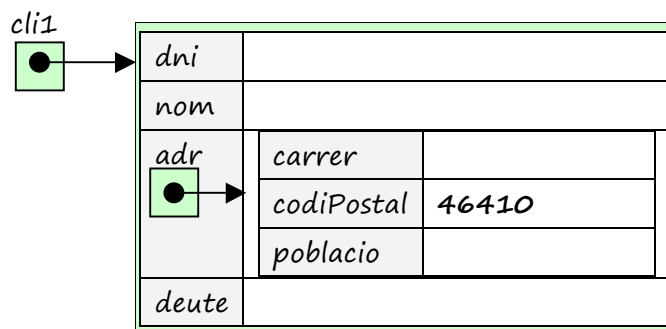
 Zona que es reserva amb el new

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

cli1.adr.codiPostal = 46410;

és com fer:

cli1→adr→codiPostal = 46410;



## Exercicis

5. Amplia el projecte Proves per a definir les classes corresponents a les següents estructures de dades, basant-te en les classes que ja estaven definides anteriorment. És a dir: caldrà fer ús de classes niuades. Fes-ho de la primera forma que hem vist: la que en la definició de la classe ja es reserva memòria per a la subclasse.
  - a. DadesPersonals: nom complet de persona, domicili i telèfon
  - b. CarnetAlumne: dades personals, data de naixement, data de sol·licitud i hora de sol·licitud
6. Crea l'objecte carnet1 (de la classe Carnet) i ompli'l de dades qualsevol.
7. Crea l'objecte persona2 (de la classe DadesPersonals) i ompli'l de dades. Crea l'objecte carnet2 (de la classe CarnetAlumne) i ompli'l amb l'objecte persona2 i altres dades qualsevol.
8. Afig 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 segona forma que hem vist: la que en la definició de la classe no es reserva memòria per a la subclasse.
9. Crea l'objecte cercle1 (de la classe Cercle) i ompli'l de dades qualsevol.

## 5 Vectors de registres

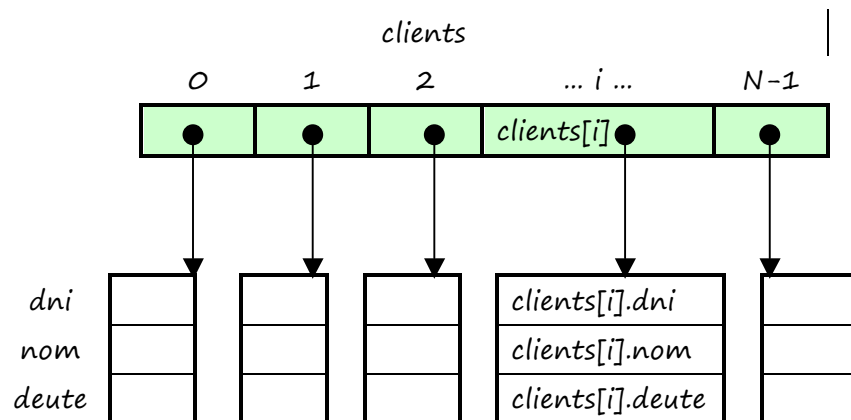
### Definició

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					
nom					
deute					

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.
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.
12. Estructura per a guardar la informació de 200 empleats: número, nom, adreça, edat, estat civil, telèfon fix i mòbil.
13. Fes un programa que demane la informació de 30 estudiants i la mostre per pantalla. Per cadascun cal saber el nom, telèfon, adreça (carrer, número i codi postal) i qualificacions: notes que ha tret en cada avaluació de cada assignatura. Hi ha 3 avaluacions i les assignatures són SIM, XAL i FPR.



14. Modifica el programa anterior. Ha de tindre un bucle amb el menú:

(A)lta alumnes  
(N)ota alumnes  
(B)uscar alumne  
(E)ixir

Les 3 primeres opcions seran crides a les següents funcions, que caldrà implementar:

- ✓ **altaAlumnes**: demanarà les dades personals de cada alumne (no les notes) i les guardarà.
- ✓ **notaAlumnes**: preguntarà quina avaluació i assignatura, i anirà demanant per teclat les notes corresponents de cada alumne (i guardant-les, clar).
- ✓ **buscarAlumne**: preguntarà pel nom d'un alumne i mostrarà totes les seues dades.

15. 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.



- a. Definix la classe *Vivenda* amb les variables: *m2*, *q\_hab*, *preu*, *nom*, *nif*
- b. Definix *edifici*, 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 l'aplicació principal amb el menú que cregues adequat.

16. 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.

17. Volem crear una aplicació que ens permeti introduir les dades de l'horari del 1r curs del cicle ASI. Per cada hora i dia cal guardar: nom del mòdul, nom del professor i número d'aula. Tria l'estructura de dades més adient per a emmagatzemar esta informació.

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

El programa tindrà el menú següent:

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

Caldrà definir i utilitzar les funcions següents:

- ✓ **omplirHorari:** s'introduirà l'horari sencer per teclat
- ✓ **mostrarHorari:** es mostrarà un submenú (\*) per a triar la dada a mostrar (Mòdul/Professor/Aula). Després de triar l'opció escollida, es mostrarà l'horari per pantalla amb la informació corresponent.
- ✓ **modificar:** es preguntarà pel dia de la setmana i hora (1 a 6) que es vol modificar. A continuació es mostrarà un submenú (\*) per a triar la dada a modificar (Mòdul/Professor/Aula). Es demanarà per teclat la nova dada a canviar i s'actualitzarà.

(\*) Cal també implementar la funció **triarSubmenú**. Esta funció no rep cap paràmetre d'entrada. Ha de mostrar el submenú, preguntar per teclat quina opció i retornar-la.