NF3 TIPUS DE DADES COMPOSTES

Arrays o vectors

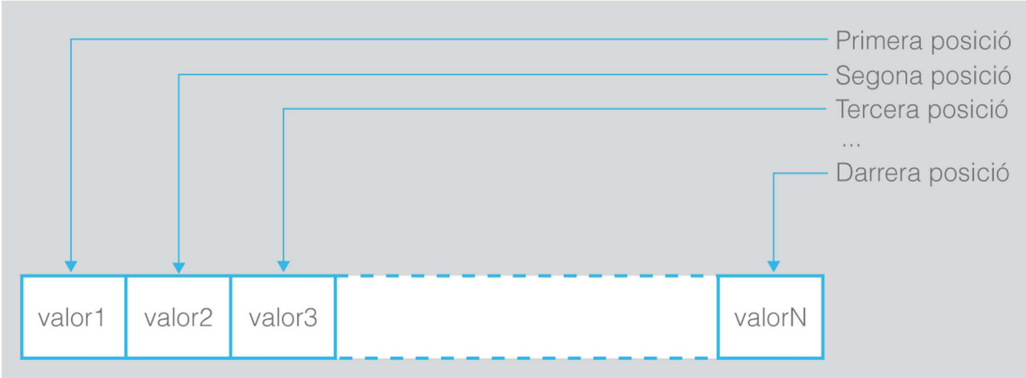
Suposeu que voleu fer un programa per analitzar les notes finals d'un conjunt d'estudiants. Les operacions per fer poden ser diverses: calcular la nota mitjana, veure quants estudiants han aprovat, suspès o han superat cert llindar de nota, fer gràfiques de rendiment, etc. A més, no es descarta que en el futur el programa es modifiqui per afegir-hi noves funcionalitats. El *quid* de la qüestió és que, per poder preveure totes aquestes possibilitats i d'altres de futures encara desconegudes, us cal disposar dins del programa del valor de cada nota individual. En un cas com aquest, tal com s'ha plantejat fins ara l'aproximació per manipular dades dins d'un programa, això implica que cada nota s'ha d'emmagatzemar dins d'una variable diferent. Si hi ha 50 estudiants, això vol dir que caldria declarar 50 variables.

Malauradament, aquesta aproximació no funciona en un cas com aquest, en què el nombre de dades per processar és relativament alt. D'una banda, imagineu-vos l'aspecte que tindrà un codi font en què cal declarar i manipular 50 variables per fer tota mena de càlculs. Les expressions per fer càlculs entre elles, com simplement fer la mitjana (sumar-les totes i dividir pel nombre de valors), serien enormes. D'altra banda, què passa si cal processar les notes de dos-cents o mil estudiants? És factible declarar mil variables? Pitjor encara: què passa si el nombre d'estudiants de cada curs és totalment diferent? En cada curs el nombre de variables usades no encaixarà amb el d'estudiants i serà necessari modificar el codi font tenint en compte el nombre exacte de valors per tractar i tornar a compilar.

Ja a simple vista es pot apreciar que tot plegat és inviable. Cal un sistema flexible per emmagatzemar un nombre arbitrari de valors de manera que aquests siguin fàcils de manipular. Encara més, ha de ser possible que el nombre de valors emmagatzemats pugui ser diferent per a cada execució del programa. Per resoldre aquesta problemàtica, els llenguatges de programació d'alt nivell ofereixen el tipus de dada compost *array*, o taula.

Un **tipus de dada compost** és aquell que permet emmagatzemar més d'un valor dins d'una única variable. En el cas de l'**array**, aquest permet emmagatzemar, en forma de seqüència, una quantitat predeterminada de valors pertanyents al **mateix tipus** de dades.

Per tal de diferenciar els diferents valors emmagatzemats, l'*array* gestiona el seu contingut d'acord amb posicions que segueixen un ordre numèric: el valor emmagatzemat a la primera posició, a la segona, a la tercera, etc. A efectes pràctics, es pot considerar que cada posició individual es comporta exactament igual que una variable del tipus de dada triat. Tant es pot consultar el valor com emmagatzemar-hi dades. Aquest comportament s'esquematitza a la imatge següent.



Declaració i inicialització d'arrays

Tal com passa amb les variables, un array també te un tipus i li podem donar un valor inicial

- Hem de decidir quantes posicions tindrà el vector i no canviarà al llarg del programa.
- Cada posició del vector es comporta com una variable i pot convenir tenir inicialitzada cada posició a un valor concret.
- No podem barrejar diferents tipus de dades dins un array.

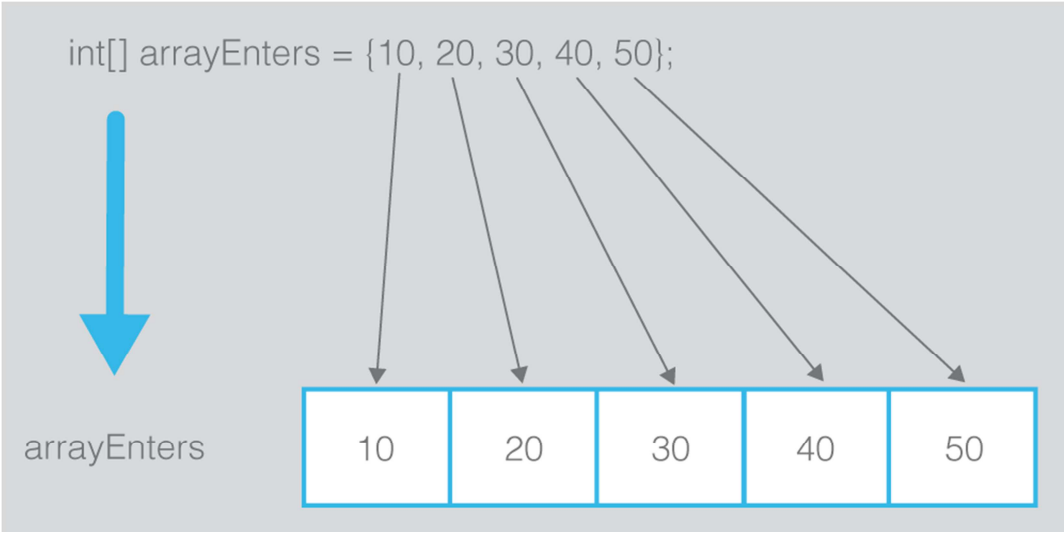
Veiem alguns exemples:

INICIALITZACIÓ A UN VALOR CONCRET

```
paraulaClauTipus[] identificadorVariable = {valor1, valor2, ... , valorN};
```

Per exemple, per declarar un *array* de 5 posicions en què es poden emmagatzemar valors de tipus enter, cadascuna amb els valors inicials 10, 20, 30, 40 i 50, respectivament, es faria de la manera següent. La mida d'aquest *array* seria 5.

```
int[] arrayEnters = {10, 20, 30, 40, 50};
```



INICIALITZACIÓ A UN VALOR PER DEFECTE

Moltes vegades les dades que es volen emmagatzemar depenen de la lectura d'una entrada (per exemple, per teclat), o bé són resultat de càlculs que són més còmodes de dur a terme de

manera automatitzada dins del codi de programa (per exemple, una llista de 1.000 nombres primers). En aquest cas, no hi ha un valor inicial concret que calgui assignar a les posicions de l'*array*, ja que els valors s'emmagatzemaran *a posteriori*.

En aquest cas, una altra manera, més simple i habitual, de declarar i inicialitzar-lo, és assumint que totes les posicions prenen automàticament un valor per defecte, que és 0 per a les dades de tipus numèric i els caràcters, i false per a booleans. Més endavant ja assignareu a cada posició el valor que vulgueu.

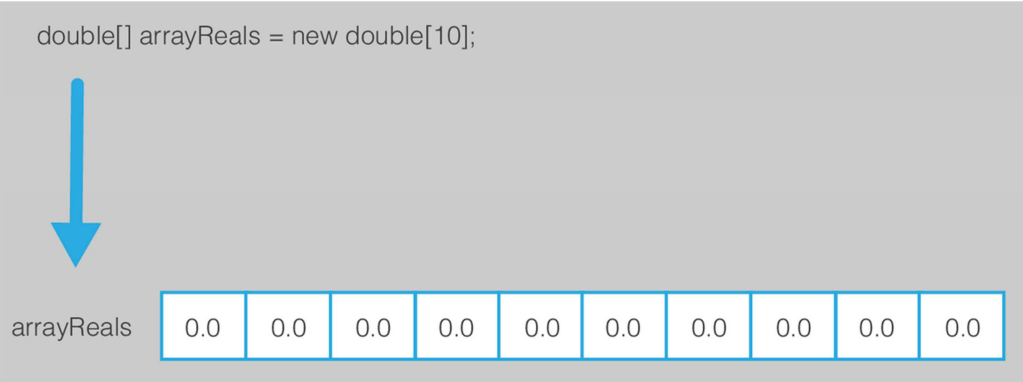
```
paraulaClauTipus[] identificadorVariable = new paraulaClauTipus[mida];
```

Per exemple, per declarar un *array* de 10 posicions en què es poden emmagatzemar valors de tipus real, tots inicialment amb el valor 0, es faria de la manera següent:

```
double[] arrayReals = new double[10];
```

Que seria equivalent a

```
double[] arrayReals = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```



Manipulació de dades dins l'array

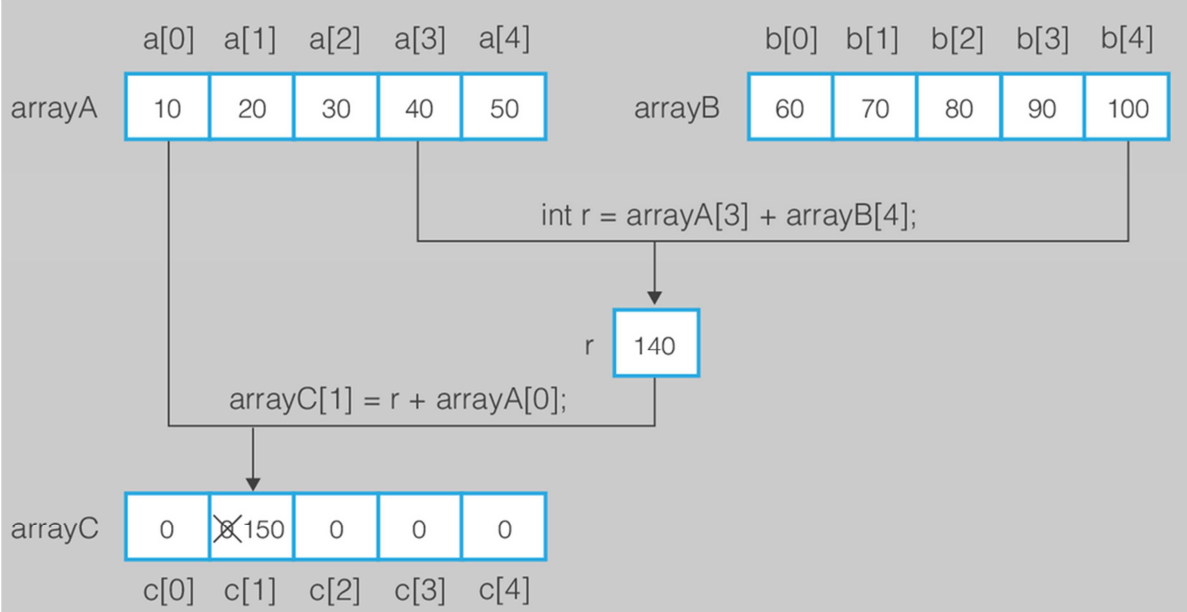
Sempre que es vulgui fer alguna operació amb les dades emmagatzemades dins d'*arrays* cal manipular-les de manera individual, posició per posició. En aquest aspecte, cada posició d'un *array* té exactament el mateix comportament que una variable tal com les heu estudiades fins ara. Les podeu aprofitar tant per emmagatzemar dades com per llegir-ne el contingut.

Per tal de distingir entre les diferents posicions d'un *array*, cadascuna té assignat un **índex**, un valor enter que n'indica l'ordre dins de l'estructura. Sempre que us vulgueu referir a una de les posicions, n'hi ha prou d'usar l'identificador de l'*array* juntament amb l'índex de la posició buscada entre claudàtors []. La sintaxi exacta és:

```
identificadorArray[índex]
```

En el cas del Java, aquests van de 0, per a la primera posició, a (mida − 1), per a la darrera. Per exemple, per accedir a les posicions d'un *array* de cinc posicions usaríeu els índexs 0, 1, 2, 3 i 4.

```java
int[] arrayA = {10, 20, 30, 40, 50};
int[] arrayB = {50, 60, 70, 80, 100};
int[] arrayC = new int[5];
//La variable de tipus enter "r" valdrà 140, 40 + 100.
int r = arrayA[3] + arrayB[4];
//La segona posició (índex 1) d'"arrayC" valdrà 150, 140 + 10.
arrayC[1] = r + arrayA[0];
```



És molt important que en accedir a un *array* s'utilitzi un índex correcte, d'acord amb el rang admissible. Si s'usa un valor no admissible, com per exemple un valor negatiu o un valor igual o superior a la seva mida, hi haurà un error d'execució. Concretament, es produirà una IndexOutOfBoundsException. Si això succeeix, podeu tenir la certesa que el programa té una errada i l'haureu de repassar i corregir. Per tant, els codi següent seria incorrecte:

```java
//"arrayA" té mida 5.
//Els índexs vàlids van de 0 a (mida - 1), que és 4.
int[] vectorD = {10, 20, 30, 40, 50};
//S'assigna un valor a una posició incorrecta (índex 5).
arrayA[5] = 60;
//L'índex 6 és invàlid, no es pot consultar el valor.
int r = vectorD[2] + vectorD[4] + vectorD[6];
```

Per tal d'ajudar-vos en la tasca de controlar si un índex concret està dins del rang admissible per a una variable de tipus *array* concreta, Java disposa d'una eina auxiliar, l'atribut length (llargària). La seva sintaxi és la següent:

```
identificadorArray.length
```

El resultat d'aquesta instrucció és equivalent a avaluar una expressió que com a resultat obté la mida de l'*array* anomenat identificadorArray. Ara bé, recordeu que length us dóna la mida, però el rang d'índexs vàlid és 0 … (mida − 1). Per exemple:

```
int[] arrayF = {10, 20, 30, 40, 50};
//"length" us diu la mida.
//Recordeu que l'índex màxim és (mida - 1).
if (index < arrayF.length) {
	System.out.println("La posició " + index + " val " + arrayF[index]);
} else {
	System.out.println("Aquest array no té tantes posicions!");

}
```

Entrada de seqüencies de valors per teclat

Abans de continuar, val la pena veure com es poden llegir seqüències de dades entrades des del teclat en una sola línia de text, de manera que sigui senzill emmagatzemar-les dins d'un *array*. Si bé ja sabeu com llegir dades individualment des del teclat, fer-ho així pot ser una mica avorrit i molest amb vista a l'execució dels programes en què s'introdueixen moltes dades, preguntant cada valor un per un i havent de pitjar la tecla de retorn cada vegada.

En realitat, quan s'usa una instrucció lector.next… i el programa s'atura esperant que l'usuari introdueixi dades pel teclat, res no impedeix que, en lloc d'una única dada, aquest n'escrigui més d'una abans de pitjar la tecla de retorn. O sigui, una seqüència de valors. L'única condició és que cada valor individual estigui separat de la resta per almenys un espai, de manera que siguin fàcilment identificables. Quan això succeeix, tots els valors de la seqüència queden latents, pendents de lectura. Successives invocacions a noves instruccions de lectura automàticament aniran consumint aquests valors pendents, en lloc de causar una nova espera d'una entrada de l'usuari. Mentre quedin valors a la seqüència pendents de llegir, les instruccions de lectura mai no causaran una espera d'entrada de dades. Un cop la seqüència ha estat consumida totalment, llavors sí que la propera instrucció de lectura causarà que el programa es torni a aturar esperant una nova entrada de l'usuari. I així el cicle de lectura torna a començar.

Dins d'aquest procés hi ha una crida amb un comportament especial: lector.nextLine();. Quan aquesta crida s'invoca, automàticament es descarten tots els valors pendents de llegir de la seqüència actual. La propera crida a una instrucció de lectura sempre causarà que el programa s'aturi de nou i esperi una entrada de l'usuari.

Un fet important quan es llegeixen seqüències de diversos valors escrites des del teclat és que res no impedeix que hi pugui haver valors de diferents tipus de dades barrejats. Això tant pot ser per error de l'usuari en entrar les dades, com perquè el programa realment espera una seqüència amb valors de diferents tipus intercalats. En qualsevol cas, és important que abans de fer cap lectura d'una dada es comprovi si el tipus és el que correspon usant les instruccions lector.hasNext….

Per llegir múltiples valors el més senzill és fer-ho mitjançant una estructura de repetició que vagi invocant successivament les instruccions de lectura de dades. En usar aquest mecanisme cal tenir present un fet ben important. S'ha de saber exactament quan heu obtingut ja totes les dades necessàries i cal deixar llegir. Normalment hi ha dues aproximacions depenent de si es coneix la quantitat de dades exactes que cal llegir o no.

Quantitat de dades coneguda

```
import java.util.Scanner;

public class LecturaSenseValidacio {

private static final int NUM_VALORS = 10;

public static void main(String[] args) {

	Scanner lector = new Scanner(System.in);
	System.out.println("Escriu " + NUM_VALORS + " enters. Es pot fer en
diferents línies.");
		//Es llegeixen exactament NUM_VALORS valors.
	int numValorsLlegits = 0;
	while (numValorsLlegits < NUM_VALORS) {
		//Abans de llegir, comprovem si realment hi ha un enter.
		if (lector.hasNextInt()) {
			int valor = lector.nextInt();
			System.out.println("Valor " + numValorsLlegits + " llegit:
" + valor);
			numValorsLlegits++;
		} else {
			//Si el valor no és enter, es llegeix però s'ignora.
			//No s'avança tampoc el comptador.
				lector.next();
		}
	}
		//Els valors que sobrin a la darrera línia escrita es descarten.
		lector.nextLine();
		System.out.println("Ja s'han llegit " + NUM_VALORS + "
valors.");
	}
```

Quantitat de dades desconeguda

Un cas més complex és quan el nombre de valors no és conegut *a priori*, ja que pot variar en diferents execucions del programa. Quan això succeeix, una solució simple seria preguntar simplement, abans de llegir cap valor, quantes dades s'introduiran, o bé fer que el primer valor dins de la seqüència n'indiqui la longitud.

Si això no és possible, o es considera que no s'ha de fer, una altra opció és establir un valor especial que no forma part de les dades per tractar dins el programa, sinó que es considerarà com a **marca de final de seqüència**. Tan bon punt es llegeixi aquest valor, la lectura s'ha de donar per finalitzada.

El codi següent d'exemple llegeix una seqüència de valors enters de llargària arbitrària. La lectura finalitza tan bon punt es llegeix el valor -1. Compareu aquest codi amb l'exemple del cas anterior.

```java
import java.util.Scanner;

public class LectorValorsDesconeguts {
	public static final int MARCA_FI = -1;
	public static void main (String[] args) {
		Scanner lector = new Scanner(System.in);
		System.out.print("Escriu diferents valors enters.");
	System.out.println("Després del darrer valor escriu un " + MARCA_FI);
		//Es llegeixen exactament NUM_VALORS valors.
		boolean marcaTrobada = false;
		while (!marcaTrobada) {
			//Abans de llegir, comprovem si realment hi ha un enter.
			if (lector.hasNextInt()) {
				int valor = lector.nextInt();
				//És la marca de fi?
				if (valor == MARCA_FI) {
					//Sí que ho és.
					marcaTrobada = true;
				} else {
					//No. És un valor que ha de ser tractat.
					System.out.println("Valor llegit: " + valor);
				}
			} else {
			//Si el valor no és enter, es llegeix però s'ignora.
				lector.next();
			}
		}
		//Els valors que sobrin a la darrera línia escrita es descarten.
		lector.nextLine();
		System.out.println("Ja s'han llegit tots els valors.");
	}
}
```

L'esquema més simple d'emmagatzematge de dades des d'una entrada a les posicions d'un *array* és el cas en què la quantitat de dades que es llegirà és coneix prèviament. Això es deu a la important restricció que, per inicialitzar un *array*, prèviament se n'ha d'establir la mida. Aquest és un requisit indispensable, ja que en cas contrari qualsevol intent de desar-hi dades resultarà en un error d'execució.

Com a exemple, suposeu un programa per tractar les notes d'avaluació d'un conjunt d'estudiants. Per obtenir les diferents notes, aquest llegeix una seqüència, de llargària coneguda, composta de nombres reals. Amb els valors llegits s'omple completament un *array*. Compileu i executeu el codi següent per dur a terme aquesta operació. Fixeu-vos que, en aquest cas, es comproven un seguit d'errors possibles a l'entrada de les dades abans d'emmagatzemar el valor final a l'*array*: si els valors introduïts són realment de tipus real i si estan dins del rang vàlid per a una nota (entre 0 i 10).

```java
import java.util.Scanner;
//Programa que llegeix una seqüència de valors reals, de longitud coneguda.
public class LectorSequencia {
	public static void main (String[] args) {
		Scanner lector = new Scanner(System.in);
		//Llegeix la longitud de la seqüència. Comprova tots els errors.
		int nombreValors = 0;
		while (nombreValors <= 0) {
			System.out.print("Quantes notes vols introduir? ");
			if (lector.hasNextInt()) {
				nombreValors = lector.nextInt();
			} else {
				//Si no és enter, es llegeix i s'ignora.
				lector.next();
			}
		}
		//Si s'han entrat més valors, s'ignoren. Només se'n necessita
un.
		lector.nextLine();
		System.out.println("Es llegiran " + nombreValors + " valors
reals.");
		System.out.println("En pots escriure diversos en una sola línia,
separats per espais.");
		//Els desarem en un array. Ja en coneixem la mida.
		float[] arrayNotes = new float[nombreValors];
		//Cal llegir tants reals com la mida de l'array.
		//Estructura de repetició amb comptador.
		int index = 0;
		while (index < arrayNotes.length) {
			if (lector.hasNextFloat()) {
				//S'ha llegit una nota, però és vàlida (entre 0 i
10)?
				float nota = lector.nextFloat();
				if ((nota >= 0)&&(nota <= 10)) {
					arrayNotes[index] = nota;
					index++;
				}
				//Si no és vàlida, la ignorem. No s'assigna enlloc.
			} else {
				//Si no era un real, simplement llegim el valor com
una cadena de text
				//però no en fem res. Es perd.
				lector.next();
			}
		}
		//Ignorem els valors sobrants de la darrera línia.
		lector.nextLine();
		System.out.println("La seqüència llegida és:");
		for (int i = 0; i < arrayNotes.length;i++) {
			System.out.println(arrayNotes[i]);
		}
	}
}
```