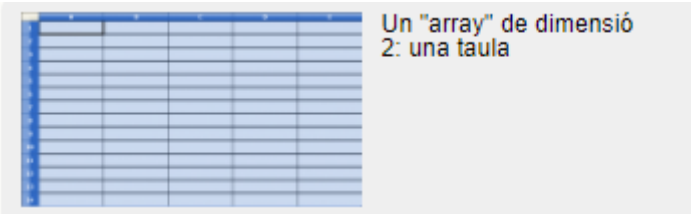
UD7. CLASSES D'EMMAGATZEMATGE I UTILITATS

7.1. Arrays o matrius

Quan es declara un array, hi ha una propietat especial, a part del seu identificador de tipus i de mida. Es tracta de la seva **dimensió**, el nombre d'índexs que cal usar per establir la posició que es vol tractar. Res no impedeix que el nombre d'índexs sigui més gran que 1.

Un **array multidimensional** és aquell en què per accedir a una posició concreta, en lloc d'usar un sol valor com a índex, s'usa una seqüència de diversos valors. Cada índex serveix com a coordenada per a una dimensió diferent.



Entre els arrays multidimensionals, el cas més usat és el dels arrays bidimensionals, de dues dimensions. Aquest és un dels més fàcils de visualitzar i és, per tant, un bon punt de partida. Com que es tracta d'un tipus d'array en què calen dos índexs per establir una posició, es pot considerar que el seu comportament és com el d'una **matriu**, **taula** o **full de càlcul**. El primer índex indicaria la **fila**, mentre que el segon indicaria la **columna** on es troba la posició a què es vol accedir.

Per exemple, suposeu que un programa per tractar notes d'estudiants ara ha de gestionar les notes individuals de cada estudiant per a un seguit d'exercicis, de manera que es poden fer càlculs addicionals: calcular la nota final segons els resultats individuals, veure l'evolució de cada estudiant al llarg del curs… En aquest cas, tota aquesta informació es pot resumir de manera eficient en una taula o en un full de càlcul, en què els estudiants es poden organitzar per files, on cada columna correspon al valor d'una prova, i la darrera columna pot ser la nota final.

Declaració d'"arrays" bidimensionals

La sintaxi per declarar i inicialitzar un array de dues dimensions és semblant a la dels unidimensionals, si bé ara cal especificar que hi ha un índex addicional. Això es fa usant un parell de claus extra, [ ], sempre que calgui especificar un nou índex.

Per declarar-ne un d'inicialitzat amb totes les posicions dels seus valors per defecte, caldria usar la sintaxi:

```
paraulaClauTipus[] identificadorVariable = new paraulaClauTipus[nombreFiles][nombreColumnes];
```

Com passava amb els arrays unidireccionals, el valor dels índexs, de les files i de les columnes, no ha de ser necessàriament un de literal. Pot ser definit d'acord amb el contingut d'una variable, diferent per a cada execució del programa. El valor del nombre de files i de columnes tampoc no ha de ser necessàriament el mateix.

La inicialització mitjançant valors concrets implica indicar tants valors com el nombre de files per columnes. Per fer-ho, primer s'enumeren els valors individuals que hi ha a cada fila de la mateixa manera que s'inicialitza un array unidimensional: una seqüència de valors separats per comes i envoltats per claudàtors, {…}. Un cop enumerades les files d'aquesta manera, aquestes s'enumeren al seu torn separades per comes i envoltades per claudàtors. Conceptualment, és com declarar un array de files, en què cada posició té un altre array de valors. Això es veu més clar amb la sintaxi, que és la següent:

```
paraulaClauTipus[][] identificadorVariable = {
                        {Fila0valor1, Fila0valor2, ... , Fila0valorN},
                        {Fila1valor1, Fila1valor2, ... , Fila1valorN},
                        ...,
                        {FilaNvalor1, FilaNvalor2, ... , FilaNvalorN}
                        };
```

Fixeu-vos com a l'inici i al final hi ha els claudàtors extra que tanquen la seqüència de files i després de cada fila hi ha una coma, per separar-les entre si. Per fer més clara la lectura, s'han usat diferents línies de text per indicar els valors emmagatzemats a cada fila, però res no impedeix escriure-ho tot en una sola línia molt llarga. De tota manera, és recomanable usar aquest format i intentar deixar espais on pertoqui de manera que els valors quedin alineats verticalment, ja que això facilita la comprensió del codi font.

Per exemple, la manera de declarar i d'inicialitzar amb valors concrets un array bidimensional de tres files per cinc columnes de valors de tipus enter seria:

```
int[][] arrayBidi = {
                    {1 ,2 ,3 ,4 ,5 },
                    {6 ,7 ,8 ,9 ,10},
                    {11,12,13,14,15}
                    };
```

Des del punt de vista d'una taula o matriu, la primera fila seria {1, 2, 3, 4, 5}, la segona {6 ,7 ,8 ,9 ,10} i la tercera {11,12,13,14,15}, tal com mostra la taula.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

En usar aquest tipus d'inicialització cal assegurar-se que les mides de totes les files siguin exactament iguals (que tinguin el mateix nombre de valors separats entre comes), ja que cal garantir que totes les files tenen el mateix nombre de columnes.

Esquemes d'ús d'"arrays" bidimensionals

L'accés a les posicions d'un array bidimensional és pràcticament idèntic a l'unidireccional, però tenint en compte que ara hi ha dos índexs per preveure referents a les dues coordenades.

```
identificadorVariable[índexFila][índexColumna]
```

La taula fa un resum de com s'accediria a les diferents posicions d'un array de 4 * 5 posicions. En qualsevol cas, en accedir-hi, sempre cal mantenir present que cada índex mai no pot ser igual o superior al nombre de files o de columnes, segons correspongui. En cas contrari, es produeix un error.

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

Com que ara hi ha dos índexs, tots els esquemes fonamentals per treballar amb arrays bidimensionals es basen en dues estructures de repetició imbricades, una per tractar cada fila i l'altra per tractar cada valor individual dins la fila.

En el cas d'arrays bidimensionals, l'atribut length té un comportament una mica especial, ja que ara hi ha dos índexs. Per tal entendre'l, cal recordar que per inicialitzar l'array el que es fa és enumerar una llista de files, dins les quals hi ha els valors emmagatzemats realment. Per tant, el que diu l'atribut és el nombre de files que hi ha. Per saber el **nombre de valors d'una fila** heu de fer:

```
identificadorVariable[índexFila].length
```

Per exemple, si la taula correspon a una variable anomenada arrayBidi, arrayBidi.length avalua a 4 (hi ha quatre files) i arrayBidi[0].length, arrayBidi[1].length, etc. tots avaluen a 5 (hi ha 5 columnes).

Inicialització procedural

Novament, es pot donar el cas en què vulgueu assignar a cada posició valors que cal calcular prèviament i que, per tant, calgui anar assignant un per un a totes les posicions de l'array bidimensional.

A mode d'exemple, compileu i executeu el codi font següent, que emmagatzema a cada posició la suma dels índexs d'un array bidimensional de dimensions arbitràries i després es visualitza el resultat per pantalla, ordenat per files.

```
import java.util.Scanner;

//Un programa que inicialitza un array bidimensional.
public class InicialitzacioBidi {

	public static void main (String[] args) {

		Scanner lector = new Scanner(System.in);

		//Llegeix les files.
		int nombreFiles = 0;
		while (nombreFiles <= 0 ) {
```

```
                        System.out.print("Quantes files tindrà la taula? " );
                        if (lector.hasNextInt() ) {
                                nombreFiles = lector.nextInt();
                        } else {
                                lector.next();
                                System.out.print("Aquest valor no és correcte. " );
                        }
                }
                lector.nextLine();

                //Llegeix les columnes.
                int nombreColumnes = 0;
                while (nombreColumnes <= 0) {
                        System.out.print("Quantes columnes tindrà la taula? " );
                        if (lector.hasNextInt() ) {
                                nombreColumnes = lector.nextInt();
                        } else {
                                lector.next();
                                System.out.print("Aquest valor no és correcte. ");
                        }
                }
                lector.nextLine();

                //Inicialització amb valors per defecte.
                int[][] arrayBidi = new int[nombreFiles][nombreColumnes];

                //Observeu l'ús de l'atribut "length".
                System.out.println("Hi ha " + arrayBidi.length + " files." );
                System.out.println("Hi ha " + arrayBidi[0].length + "columnes."
);

                //Bucle per recórrer cada fila.
                //"i" indica el número de fila.
                for(int i = 0; i < nombreFiles; i++) {

                        //Bucle per recórrer cada posició dins la fila (columnes
de la fila).
                        //"j" indica el número de fila.
                        for (int j = 0; j < nombreColumnes; j++) {

                                //Valor assignat a la posició: suma dels índex de
fila i columna.
                                arrayBidi[i][j] = i + j;
                        }
                }
                //Es visualitza el resultat, també calen dos bucles.
                for(int i = 0; i < nombreFiles; i++) {
                        //Inici de fila, obrim claudàtors.
                        System.out.print("Fila " + i + " { ");
                        for (int j = 0; j < nombreColumnes; j++ ) {
                                System.out.print(arrayBidi[i][j] + " ");
                        }
                        //Al final de cada fila es tanquen claudàtors i es fa un
salt de línia.
                        System.out.println("}" );
                }

        }
}
```

Recorregut

En el cas d'arrays bidimensionals, també pot passar que sigui necessari efectuar càlculs fent un recorregut per tots els valors continguts. Novament, cal anar fila per fila, mirant totes les posicions de cadascuna.

Per exemple, suposeu que hi ha un programa en què es desa a cada fila el valor de les notes dels estudiants d'un curs. Es demana calcular la nota final de cada estudiant i emmagatzemar-la a la darrera columna, i també saber la mitjana de totes les notes finals. El codi es mostra tot seguit; analitzeu-lo i proveu que funciona. En aquest cas, per facilitar-ne l'execució, els valors de les notes estan inicialitzats per a valors concrets. En el cas de la nota final de cada estudiant, inicialment es deixa a 0 i serà el programa qui la calculi.

Ara sí: observeu atentament l'ús de l'atribut length per controlar els índexs en recórrer l'array i saber quin és el nombre de valors en una fila o el nombre de files.

```
//Un programa que calcula notes mitjanes en un array bidimensional.
public class RecorregutBidi {
	public static void main (String[] args) {
		//Dades de les notes.
		float[][] arrayBidiNotes = {
				{ 4.5f, 6f , 5f , 8f , 0f },
				{ 10f , 8f , 7.5f, 9.5f, 0f },
				{ 3f , 2.5f, 4.5f, 6f , 0f },
				{ 6f , 8.5f, 6f , 4f , 0f },
				{ 9f , 7.5f, 7f , 8f , 0f }
		};
		//Mitjana aritmètica del curs per a tots els estudiants.
		float sumaFinals = 0f;

		//Es fa tractant fila per fila, indicada per "i". Cada fila és
un estudiant.
		//"arrayBidiNotes.length" avalua el nombre de files.
		for(int i = 0; i < arrayBidiNotes.length; i++) {
			//Aquí s'acumulen les notes de l'estudiant tractat.
			float sumaNotes = 0f;

			//Tractem cada fila (cada estudiant). Cada nota la indexa
"j".
			//"arrayBidiNotes[i].length avalua el nombre de posicions
a la fila.
			for(int j = 0; j < arrayBidiNotes[i].length; j++) {
				//Estem a la darrera posició de la fila?
				if(j != (arrayBidiNotes[i].length - 1) ) {
					//Si no és la darrera posició, anem acumulant
valors.
					sumaNotes = sumaNotes + arrayBidiNotes[i][j];
				} else {
					//Si ho és, cal escriure la mitjana.
					//Hi ha tantes notes com la mida d'una fila –
1.
					float notaFinal =
sumaNotes/(arrayBidiNotes[i].length - 1);
					arrayBidiNotes[i][j] = notaFinal;
					System.out.println("L'estudiant " + i + " ha
tret " + notaFinal + ".");
```

```
                                //S'actualitza la suma de mitjanes de tots
els estudiants.
                                sumaFinals = sumaFinals + notaFinal;
                        }
                }
                //Fi del tractament d'una fila.

        }
        //Fi del tractament de totes les files.
        //Es calcula la mitjana: suma de notes finals dividit entre
nombre d'estudiants.
        float mitjanaFinal = sumaFinals / arrayBidiNotes.length;
        System.out.println("La nota mitjana del curs és " +
mitjanaFinal);
    }
}
```

Cerca

En arrays bidimensionals la cerca també implica haver de gestionar dos índexs per anar avançant per files i per columnes. Ara bé, aquest cas és especialment interessant, ja que en assolir l'objectiu cal sortir de les dues estructures de repetició imbricades. Per tant, no es pot usar simplement una sentència break, ja que només serveix per sortir d'un únic bucle. Cal tenir un semàfor que s'avaluï en tots dos bucles.

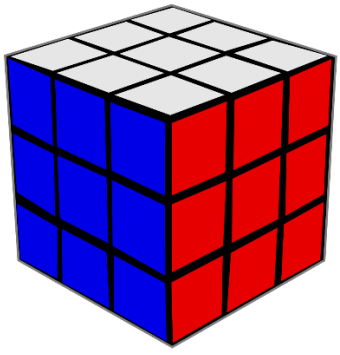
Per exemple, suposeu que el programa de gestió de notes vol cercar si algun estudiant ha tret un 0 en algun dels exercicis al llarg del curs. Caldrà anar mirant fila per fila totes les notes de cada estudiant, i quan es trobi un 0, acabar immediatament la cerca. El codi per fer-ho, basant-se exclusivament en un semàfor que diu si ja s'ha trobat el valor, seria el següent. Proveu-lo al vostre entorn de treball.

En un cas com aquest, cal anar amb molt de compte a inicialitzar i incrementar correctament l'índex per recórrer la posició de cada fila (és a dir, que mira cada columna), ja que en usar una sentència while en lloc de for no es fa automàticament. Si us despisteu i no ho feu, el valor serà incorrecte per a successives iteracions del bucle que recorre les files i el programa no actuarà correctament.

```
//Un programa que cerca un 0 entre els valors d'un array bidimensional.
public class CercaBidi {
    public static void main (String[] args) {
        //Dades de les notes.
        float[][] arrayBidiNotes = {
                    { 4.5f, 6f , 5f , 8f },
                    { 10f , 8f , 7.5f, 9.5f},
                    { 3f , 2.5f, 0f , 6f },
                    { 6f , 8.5f, 6f , 4f },
                    { 9f , 7.5f, 7f , 8f }
        };

        //Mirarem quin estudiant ha tret el 0.
        //Inicialitzem a un valor invàlid (de moment, cap estudiant té
un 0)
        //Aquest valor fa de semàfor. Si pren un valor diferent, cal
acabar la cerca.
        int estudiant = -1;

        //"i indica la fila.
        int i =0;
```

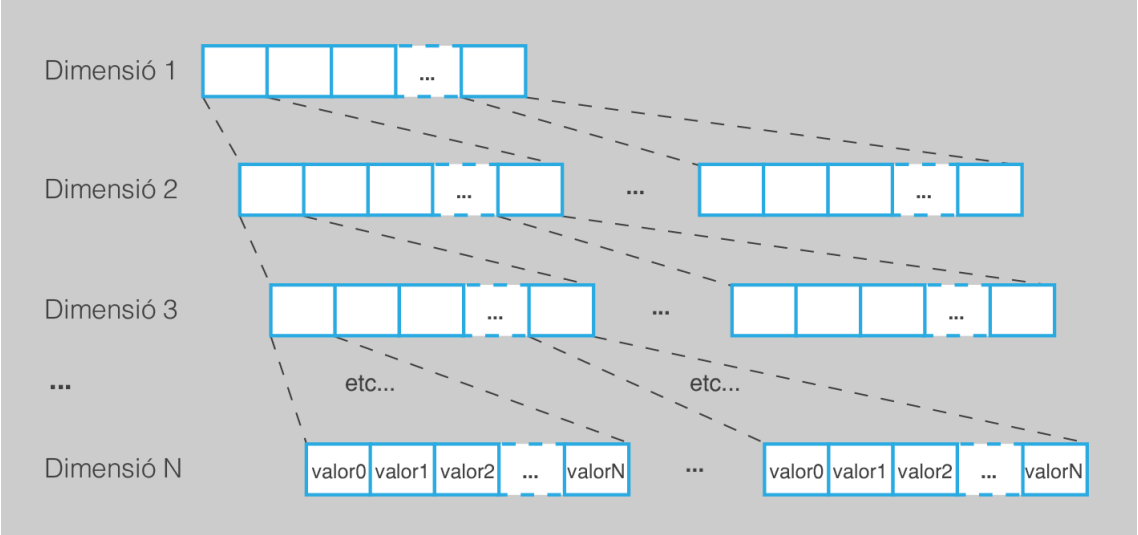
```
		//Es va fila per fila.
		//S'acaba si s'ha trobat un 0 o si ja s'ha cercat a totes les
files.
		while ((estudiant == -1)&&(i < arrayBidiNotes.length) ){
			//"j indica la "columna".
			int j =0;

			//Se cerca en una fila.
			//S'acaba si s'ha trobat un 0 o si ja s'ha cercat a totes
les posicions.
			while ((estudiant == -1)&&(j < arrayBidiNotes[i].length)
){
				//Aquesta nota és un 0?
				if (arrayBidiNotes[i][j] == 0f) {
					//L'índex que diu l'estudiant és el de la
fila.
					estudiant = i;
				}
				//S'avança a la posició següent dins de la fila.
				j++;
			}
			//Fi del tractament d'una fila.
			//S'avança a la fila següent.
			i++;
		}
		//Fi del tractament de totes les files.

		if (estudiant == -1) {
			System.out.println("Cap estudiant no té un 0.");
		} else {
			System.out.println("L'estudiant " + estudiant + " té un
0.");
		}
	}
}
```

"Arrays" de més de dues dimensions

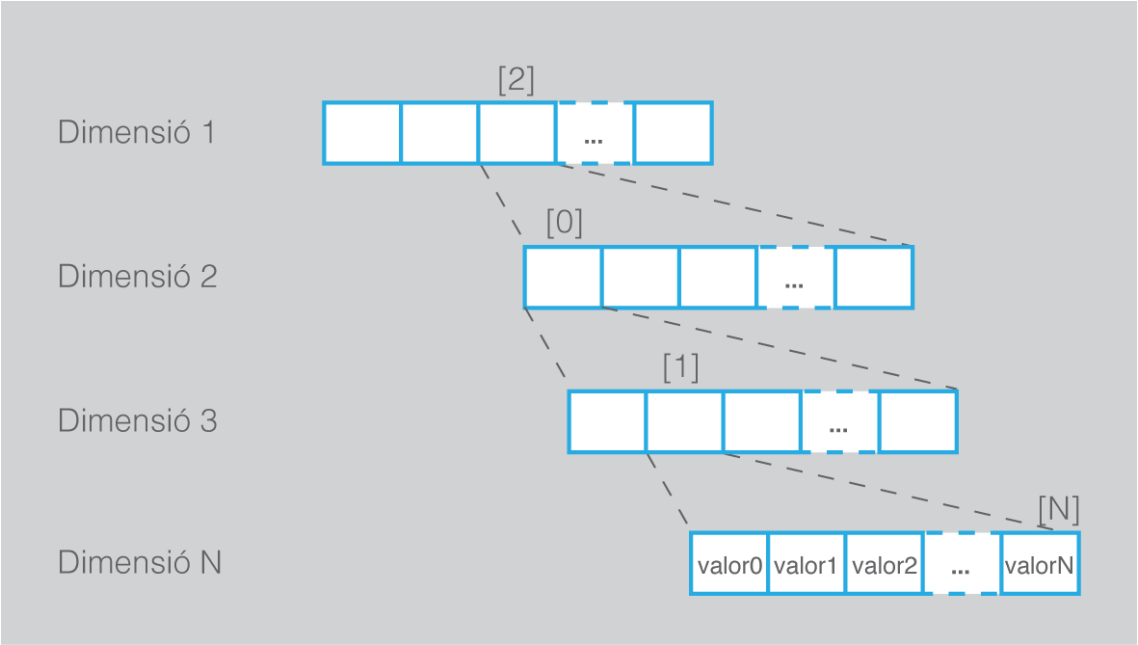


El cas dels arrays bidimensionals és el més comú, però res no impedeix que el nombre d'índexs necessaris per situar una posició sigui també més gran de dos, d'un valor arbitrari. Un array de tres dimensions encara es pot visualitzar com una estructura tridimensional en forma de cub, però dimensions superiors ja requereixen més grau d'abstracció, i per això només es presentaran, sense entrar en més detall.

Conceptualment, es pot considerar que defineixen una estructura d'arrays imbricats a diferents nivells com el que es mostra a la figura. Cada nivell correspon a una dimensió diferent.



Cada índex individual identifica una casella de l'array triat d'una dimensió determinada, de manera que el conjunt d'índexs serveix com a coordenada per establir el valor final (que ja no serà un altre array) al qual es vol accedir. La figura en mostra un exemple de quatre dimensions.



Tot i que aquesta estructura sembla força complexa, no és gaire diferent d'una estructura de carpetes i de fitxers, en què les carpetes prenen el paper dels arrays i els fitxers, el de les dades que es volen desar. Podeu tenir una única carpeta per emmagatzemar totes les fotos, que seria el cas d'un únic nivell d'array (o sigui, els arrays tal com s'han vist fins ara). Però també les podríeu organitzar en una carpeta que al seu torn conté altres carpetes, dins les quals es desen diferents fotos. Això seria equivalent a disposar de dos nivells d'arrays. I així podríeu anar creant una estructura de carpetes amb tants nivells com volguéssiu. Per triar un fitxer n'hi ha prou a saber en quin ordre cal anar obrint les carpetes fins arribar a la carpeta final, on ja només us cal l'ordre del fitxer.

Estenem les restriccions que compleixen els arrays bidireccionals al cas dels arrays multidimensionals:

- En una mateixa dimensió, tots els arrays tindran exactament la mateixa mida.
- El tipus de dada que es desa a les posicions de la darrera dimensió serà el mateix per a tots. Per exemple, no es poden tenir reals i enters barrejats.

En llenguatge Java, per afegir noves dimensions per sobre de la segona, cal anar afegint claus [ ] addicionals a la declaració, una per a cada dimensió.

```
paraulaClauTipus[]...[] identificadorVariable = new paraulaClauTipus[midaDim1]...[midaDimN];
```

Per exemple, per declarar amb valors per defecte un array de quatre dimensions, la primera de mida 5, la segona de 10, la tercera de 4 i la quarta de 20, caldria la instrucció:

```
...
int[][][][] arrayQuatreDimensions = new int[5][10][4][20];
...
```

L'accés seria anàleg als arrays bidimensionals, però usant quatre índexs envoltats per parells de claus.

```
...
int valor = arrayQuatreDimensions[2][0][1][20];
...
```

En qualsevol cas, es tracta d'estructures complexes que no se solen usar, excepte en casos molt específics.

7.2 Cadenes

7.2.1 La classe String

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

La classe String representa cadenes de caràcters. Tots els literals de cadena dels programes Java, com ara "abc", s'implementen com a instàncies d'aquesta classe.

Les cadenes són constants; **els seus valors no es poden canviar després de crear-los**. Els buffers de cadenes admeten cadenes mutables. Com que els objectes String són immutables, es poden compartir. Per exemple:

```
String str = "abc";
```

és equivalent a:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Aquí teniu alguns exemples més de com es poden utilitzar les cadenes:

```
System.out.println("abc");
```

```
String cde = "cde";

System.out.println("abc" + cde);

String c = "abc".substring(2,3);

String d = cde.substring(1, 2);
```

La classe String inclou mètodes per examinar caràcters individuals de la seqüència, per comparar cadenes, per cercar cadenes, per extreure subcadenes i per crear una còpia d'una cadena amb tots els caràcters traduïts a majúscules o minúscules. L'assignació de majúscules i minúscules es basa en la versió estàndard Unicode especificada per la classe Character.

El llenguatge Java proporciona suport especial per a l'operador de concatenació de cadenes (+) i per a la conversió d'altres objectes en cadenes. La concatenació de cadenes s'implementa mitjançant la classe StringBuilder (o StringBuffer) i el seu mètode append. Les conversions de cadena s'implementen mitjançant el mètode toString, definit per Object i heretat per totes les classes de Java.

Llevat que s'indiqui el contrari, passar un argument nul a un constructor o mètode d'aquesta classe provocarà una excepció NullPointerException.

Mètodes interessants:

- char charAt(int index) Returns the char value at the specified index.
- int compareTo(String anotherString) Compares two strings lexicographically.
- boolean equals(Object anObject) Compares this string to the specified object.
- int indexOf(int ch) Returns the index within this string of the first occurrence of the specified character.
- boolean isEmpty() Returns true if, and only if, length() is 0.
- int length() Returns the length of this string.
- String replace(char oldChar, char newChar) Returns a string resulting from replacing all occurrences of oldChar in this string with newChar.
- String[ ] split(String regex) Splits this string around matches of the given regular expression.
- boolean startsWith(String prefix) Tests if this string starts with the specified prefix.
- String substring(int beginIndex) Returns a string that is a substring of this string.
- char[ ] toCharArray() Converts this string to a new character array.
- String toLowerCase() Converts all of the characters in this String to lower case using the rules of the default locale.
- String toUpperCase() Converts all of the characters in this String to upper case using the rules of the default locale.
- static String valueOf(char c) Returns the string representation of the char argument.

7.2.2 La classe StringBuffer

https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html

public final class StringBuffer extends Object implements Serializable, CharSequence

Un StringBuffer és com una String, però es pot modificar. En qualsevol moment conté una seqüència particular de caràcters, però la longitud i el contingut de la seqüència es poden canviar mitjançant determinades cridades a mètodes.

Les operacions principals en un StringBuffer són els mètodes d'**afegir** i **inserir**, que es sobrecarreguen per acceptar dades de qualsevol tipus. Cadascun converteix efectivament una dada determinada en una cadena i després afegeix o insereix els caràcters d'aquesta cadena a l'StringBuffer. El mètode **append** sempre afegeix aquests caràcters al final de l'StringBuffer; el mètode *insert* afegeix els caràcters en un punt especificat.

Per exemple, si z es refereix a un StringBuffer amb el contingut actual "inici", aleshores la crida al mètode z.append("ar") faria que l'StringBuffer contingui "iniciar", mentre que z.insert(2, " d") alteraria l'StringBuffer perquè contingués "indici".

En general, si sb es refereix a una instància d'un StringBuffer, aleshores sb.append(x) té el mateix efecte que sb.insert(sb.length(), x).

Cada StringBuffer té una capacitat. Mentre la longitud de la seqüència de caràcters continguda a l'StringBuffer no superi la capacitat, no és necessari assignar més memòria intermèdia interna. Si el buffer intern es desborda, automàticament es fa més gran.

Llevat que s'indiqui el contrari, passar un argument nul a un constructor o mètode d'aquesta classe provocarà una excepció NullPointerException.

A partir de la versió JDK 5, aquesta classe s'ha complementat amb una classe equivalent dissenyada per a l'ús d'un sol fil, StringBuilder. La classe StringBuilder s'ha d'utilitzar generalment amb preferència a aquesta, ja que admet totes les mateixes operacions però és més ràpida, ja que no realitza cap sincronització.

Constructors:

```
StringBuffer()
Constructs a string buffer with no characters in it and an initial capacity
of 16 characters.
StringBuffer(CharSequence seq)
Constructs a string buffer that contains the same characters as the specified
CharSequence.
StringBuffer(int capacity)
Constructs a string buffer with no characters in it and the specified initial
capacity.
StringBuffer(String str)
Constructs a string buffer initialized to the contents of the specified string.
```

Mètodes més útils:

- `StringBuffer append(CharSequence s)` Appends the specified CharSequence to this sequence.
- `int capacity()` Returns the current capacity.
- `char charAt(int index)` Returns the char value in this sequence at the specified index.
- `StringBuffer delete(int start, int end)` Removes the characters in a substring of this sequence.
- `StringBuffer deleteCharAt(int index)` Removes the char at the specified position in this sequence.

- `int indexOf(String str)` Returns the index within this string of the first occurrence of the specified substring.
- `StringBuffer insert(int offset, String str)` Inserts the string into this character sequence.
- `Int length()` Returns the length (character count).
- `StringBuffer reverse()` Causes this character sequence to be replaced by the reverse of the sequence.
- `CharSequence subSequence(int start, int end)` Returns a new character sequence that is a subsequence of this sequence.
- `String substring(int start , int end)` Returns a new String that contains a subsequence of characters currently contained in this character sequence.

7.2.3 Arrays de cadenes

La matriu de cadenes de Java s'utilitza per contenir un nombre fix de cadenes. L'array de cadenes és molt comú en programes Java simples, especialment entre els principiants a Java i per provar alguns escenaris específics.

- L'array de cadenes de Java és bàsicament un array d'objectes.
- Hi ha dues maneres de declarar un array de cadenes: declaració sense mida i declaració amb mida.
- Hi ha dues maneres d'inicialitzar l'array cadenes: en el moment de la declaració, emplenant els valors després de la declaració.
- Podem fer diferents tipus de processament a l'array de cadenes com ara iteració, ordenació, cerca, etc.

Declaració d'array de cadenes de Java

A continuació, el fragment de codi mostra diferents maneres de declarar la matriu de cadenes a Java.

```
String[] strArray; //declara sense mida
String[] strArray1 = cadena nova[3]; //declara amb la mida
```

Tingueu en compte que també podem escriure una matriu de cadenes com a `String strArray[]`, però a dalt es mostra la manera estàndard i recomanada. També al codi anterior, `strArray` és nul, mentre que el valor `strArray1` és [nul, nul, nul].

Inicialització de la matriu de cadenes de Java

Vegem diferents maneres d'inicialitzar la matriu de cadenes a Java.

```
// Inicialització en línia
String[] strArray1 = new String[] {"A","B","C"};
String[] strArray2 = {"A","B","C"};

//inicialització després de la declaració
String[] strArray3 = cadena nova[3];
strArray3[0] = "A";
strArray3[1] = "B";
strArray3[2] = "C";
```

Les tres matrius de cadenes tindran els mateixos valors. Tanmateix, si els crideu al mètode equals, tornarà false.

```
System.out.println(strArray1.equals(strArray2)); // fals
System.out.println(Arrays.toString(strArray1).equals(Arrays.toString(strArray
2)));// true
```

El motiu és que la matriu és Objectes i la classe Object implementa el mètode equals() com a continuació.

```
public boolean equals(Object obj) {
        return (this == obj);
    }
```

La segona declaració és certa perquè quan es converteix a String, els seus valors són iguals i la implementació del mètode String class equals() comprova els valors. Per obtenir més detalls, consulteu la documentació de l'API de la classe String.

Iterant sobre l'array de cadenes de Java

Podem iterar sobre una matriu de cadenes utilitzant java for loop o java foreach loop.

```
String[] strArray2 = {"A","B","C"};
per (int i = 0; i < strArray2.length; i++) {
      System.out.print(strArray2[i]);
}

for (String str: strArray2) {
      System.out.print(str);
}
```

Cercar una cadena a l'array String

Podem utilitzar for loop per cercar una cadena a la matriu, a continuació es mostra un exemple senzill per això.

```
public class JavaStringArrayExample {

      public static void main(String[] args) {
            String[] strArray = { "A", "B", "C" };

            boolean found = false;
            int index = 0;
```

```
                String s = "B";
                for (int i = 0; i < strArray.length; i++) {
                        if(s.equals(strArray[i])) {
                                index = i; found = true; break;
                        }
                }
                if(found)
                        System.out.println(s +" found at index "+index);
                else
                        System.out.println(s +" not found in the array");

        }

}
```

Observeu l'ús de la paraula clau break per sortir del bucle tan bon punt hem trobat la cadena.

Ordenació d'arrays de cadenes de Java

Podem implementar el nostre propi algorisme d'ordenació, o podem utilitzar el mètode d'ordenació de classes Arrays.

```
String[] vowels = {"a","i","u","e","o"};

System.out.println("Before sorting "+Arrays.toString(vowels));

Arrays.sort(vowels);

System.out.println("After sorting "+Arrays.toString(vowels));
```

La sortida del fragment de codi anterior serà:

```
Before sorting [a, i, u, e, o]
After sorting [a, e, i, o, u]
```

Tingueu en compte que String implementa una interfície Comparable, de manera que funciona per a l'ordenació natural. En cas que vulgueu ordenar d'una altra manera, podeu utilitzar el mètode sobrecarregat Arrays.sort() passant un comparador. Obteniu informació sobre aquestes tècniques d'ordenació a Comparable i Comparator a Java.

Converteix String en String Array

Podem convertir String en matriu de cadenes utilitzant el seu mètode split(). És útil quan obteniu una única cadena com a entrada amb valors separats mitjançant un caràcter delimitador.

```
String str = "a,e,i,o,u";
String[] vowels = str.split(",");
System.out.println(Arrays.toString(vowels)); //[a, e, i, o, u]
```

Converteix String Array en String

Podem utilitzar el mètode Arrays.toString() per convertir la matriu String a String. Tingueu en compte que la matriu no implementa el mètode toString(), de manera que si intenteu obtenir la seva representació de cadena, haureu de confiar en la classe Arrays, o bé escriure el vostre propi codi personalitzat.

```
String[] vowels = { "a", "e", "i", "o", "u" };
System.out.println(vowels);
System.out.println(Arrays.toString(vowels));
```

La sortida serà com a continuació.

```
[Ljava.lang.String;@3d04a311
[a, e, i, o, u]
```

Exercici 7.1: El joc del penjat.

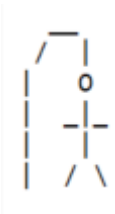
Desenvolupar en Java el joc del penjat amb les següents regles:

- S'ha de desenvolupar utilitzant la programació orientada a objectes
- La paraula a endevinar s'escollirà a l'atzar d'una llista de paraules emmagatzemades dins qualque estructura interna de dades.
- El joc mostrarà per una banda la llargària de la paraula amb les lletres que s'han encertat momentàniament, i per l'altra una llista de lletres fallades i una representació del dibuix del penjat que es pot fer utilitzant un array bidimensional per dibuixar cada tram.

Per exemple:

Paraula: - e - - a -

Errors: f m i



- el programa finalitzarà quan s'endevini tota la paraula i el jugador guanyi o s'hagi completat el dibuix del penjat amb la qual cosa el jugador haurà perdut.

7.3 Algoritmes d'ordenació

7.3.1 Inserció directa

Descripció de l'algorisme:

- Suposam una llista de k elements: 6 5 3 1 8 7 2 4
- Agafam el segon element (5): 6 _ 3 1 8 7 2 4

- Desplaçam tots els elements anteriors fins a trobar un valor menor que el seleccionat o fins que s'acaben els elements _ 6 3 1 8 7 2 4
- Insertam el valor seleccionat: 5 6 3 1 8 7 2 4
- Ara agafam el següent element no ordenat (3): 5 6 _ 1 8 7 2 4
- Desplaçam tots els elements anteriors fins a trobar-ne un de més petit o fins a acabar la llista: _ 5 6 1 8 7 2 4
- Insertam el valor seleccionat: 3 5 6 1 8 7 2 4
- ...

Animació explicativa: https://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n#/media/Archivo:Insertion-sort-example-300px.gif

Exercici 7.2: Implementar en Java l'algorisme d'ordenació per inserció directa

7.3.2 Selecció directa

Descripció de l'algorisme

- Iniciam amb una llista desordenada: 8 5 2 6 9 3 1 4 0 7
- Buscar el mínim element de la llista (0)
- Intercanviar-ho amb el primer (8): **0** 5 2 6 9 3 1 4 **8** 7
- Buscar el següent mínim a la resta de la llista (1)
- Intercanviar-ho amb el segon: 0 **1** 2 6 9 3 **5** 4 8 7

I en general:

- Buscar el mínim element entre una posició i i el final de la llista
- Intercanviar el mínim amb l'element de la posició i

Animació explicativa: https://commons.wikimedia.org/wiki/File:Selection-Sort-Animation.gif

Exercici 7.3: Implementar en Java l'algorisme d'ordenació per selecció directa

7.3.3 Intercanvi directe (bombolla)

Descripció de l'algorisme:

Es comparen dos elements consecutius i si el de la dreta és més petit que el de l'esquerra, s'intercanvien les posicions. Ara es torna a comparar amb el següent i es repeteixen les comparacions i els intercanvis fins arribar al final de la llista.

A cada iteració ens asseguram que el valor més gran queda ubicat al final de la llista.

Si tenim una llista amb n elements repetirem l'algorisme n-1 vegades.

Animació: https://es.wikipedia.org/wiki/Ordenamiento_de_burbuja#/media/Archivo:Bubble-sort-example-300px.gif

Exercici 7.4: Implementar en Java l'algorisme d'ordenació per intercanvi directa (mètode de la bombolla)

Exercici 7.5: Crear una classe OrdenarArray que permeti ordenar un array seleccionant l'algoritme que volem utilitzar per fer-ho.

7.4 Recursivitat

Ordenació ràpida (Quicksort)

L'algorisme bàsic del mètode quicksort consisteix en agafar qualsevol element de la llista que anomenarem **pivot.** Després hem d'ubicar els altres elements de manera que els més petits que el pivot quedin a l'esquerra d'ell i els majors a la dreta. Per fer això recorrerem la llista amb dos índexs 'i' i 'j', 'i' començarà de l'índex 0 (esquerra) i 'j' de l'índex length-1 (dreta) i s'aniran movent un cap amunt i l'altra cap avall fins que trobin un valor que estigui mal col·locat, es a dir un valor major que el pivot a l'esquerra d'ell i un valor major que el pivot a la dreta d'ell. En aquest punt, s'intercanviaran els dos i s'actualitzen els índexs.

Aquest procés es repetirà fins que els índexs 'i' i 'j' es creuin, es a dir que i passi a ser més gran que j (i > j). Quan això passi tindrem a l'esquerra del pivot tots els números menors que ell i a la dreta tots els majors. Aleshores el que hem de fer és aprofitar-nos de la **recursivitat** i tornar a cridar el mateix programa per aplicar el mateix procés a la part esquerra (els valors menors) i a la part dreta (els valors majors) sempre i quant hi hagi més d'un element a les subllistes.

Aquí teniu un pseudocodi que resol aquest mètode d'ordenació:

```
function quicksort(array,first_index, last_index)
    if length(array) > 1
        pivot := select any element of array
        left := first index of array
        right := last index of array
        while left ≤ right
            while array[left] < pivot
                left := left + 1
            while array[right] > pivot
                right := right - 1
            if left ≤ right
                swap array[left] with array[right]
                left := left + 1
                right := right - 1
        quicksort(array from first index to right)
        quicksort(array from left to last index)
```

En aquest enllaç podeu veure un exemple:
https://commons.wikimedia.org/wiki/File:Quicksort-example.gif

Exercici 7.6: Implementar l'algorisme de quicksort en Java.

El problema de les vuit reines

El problema de les vuit reines és un passatemps que consisteix a posar vuit reines al tauler d'escacs sense que s'amenacin. Va ser proposat per l'escaquista alemany Max Bezzel en 1848. En el joc dels escacs la reina amenaça a aquelles peces que es trobin a la seva mateixa fila, columna o diagonal. El joc de les 8 reines consisteix a posar sobre un tauler d'escacs vuit reines sense que aquestes s'amenacin entre elles. Per resoldre aquest problema es pot fer servir un esquema recursiu.

Per trobar la solució anirem col·locant cada reina en una columna diferent i comprovant si la reina que acabam de col·locar entra en conflicte amb qualcuna de les que ja tenim col·locades.

Hem de programar un mètode que comprovi si amb un cert número de reines hi ha conflicte (check(n)), que serà el que s'anirà cridant recursivament cada vegada amb una reina més. En els cas de 1 reina no hi haurà possibilitat de conflicte però en el cas de 2 ja si. Quan haguem arribat a cridar aquest mètode amb 8 reines i no ens doni conflicte haurem trobat una solució.

Aquest mètode check, anirà col·locant la nova reina en la columna actual i en cadascuna de les fileres del tauler i jutjarà si hi ha conflicte o no cridant a un altra mètode (judge(n)). En cas de que judge torni una resposta positiva indicant que no hi ha conflicte es pot cridar a check amb un nombre més de reines, aprofitant d'aquesta manera la recursivitat.

Aquest seria un possible esquema dels dos mètodes principals:

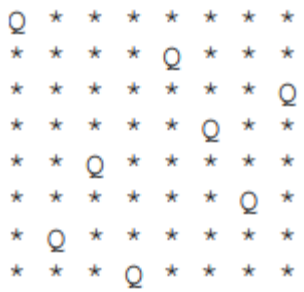
```
private void check(int n){
        if(n==totalQueens){
            imprimir solució;
            return;
        }

        for (int i = 0; i < totalQueens; i++) {
            ubicar la reina actual (n) a la fila i;
            jutjar si hi ha conflicte i si no n'hi ha crida a check amb una
reina més;
            if(judge(n){
                check(n+1);
            }
        }
    }


private boolean judge(int n){
        for (int i = 0; i < n; i++) {
            if(la posició n està en línea o en la diagonal de i)){
                return false;
            }
        }
        return true;
}
```

Exercici 7.7: Implementar l'algorisme de resolució del problema de les 8 reines en Java.

El programa ha de començar amb una crida al mètode check(0); ja que els índexs aniran de 0 a 7.

El programa ha de fer feina amb un array **unidimensional** on els índexos representaran la columna i el valor emmagatzemat representarà la fila a la qual hem ubicat la reina. Per exemple la combinació representada en aquesta imatge seria un array amb els següents valors:

[0, 6, 4, 7, 1, 3, 5, 2]



Per a cada solució trobada s'ha d'imprimir el tauler d'escacs com quedarien les reines ubicades en una imatge similar a l'anterior.

7.5 Wrappers

De vegades és molt convenient poder tractar les dades primitives (int, boolean, etc.) com a **objectes**. Per fer això, l'API de Java incorpora les classes embolcalls (wrapper class), que no són més que dotar les dades primitives amb un embolcall que permeti tractar-les com a objectes. Per exemple podríem definir una classe embolcall per als sencers, de forma força senzilla, amb:

```
public class Enter {
    private int valor;

    Enter(int valor) {
        this.valor = valor;
    }

    int intValue() {
        return valor;
    }
}
```

Les classes embolcall existents són:

- **Byte** per a byte.
- **Short** per a short.
- **Integer** per a int.
- **Long** per a long.
- **Boolean** per a boolean
- **Float** per a float.
- **Double** per a double
- **Character** per a char.

Un dels mètodes més útils d'aquestes classes és el que permet transformar un valor numèric que tinguem emmagatzemat en forma de String. Per exemple Integer.parseInt ("34") o Long.parseLong ("23000");

També son útils els mètodes per transformar un format en un altra. Per exemple: Long.intValue(29L) o Long.shortValue(29L).

Totes les classe i els seus mètodes els podeu trobar a la pàgina de referència:

https://docs.oracle.com/javase/7/docs/api/index.html?java/lang/ref/Reference.html

Alguns mètodes de la classe Integer

Constructors:

- **Integer(int value)**: Crea un nou objecte Integer amb el valor int passat per paràmetre.
- **Integer(String s)**: Crea un nou objecte Integer amb el valor String passat per paràmetro.

byte	**byteValue()** Returns the value of this `Integer` as a `byte` after a narrowing primitive conversion.
static int	**compare**(int x, int y) Compares two `int` values numerically.
int	**compareTo**(**Integer** anotherInteger) Compares two `Integer` objects numerically.
double	**doubleValue**() Returns the value of this `Integer` as a `double` after a widening primitive conversion.
boolean	**equals**(**Object** obj) Compares this object to the specified object.
float	**floatValue**() Returns the value of this `Integer` as a `float` after a widening primitive conversion.
int	**intValue**() Returns the value of this `Integer` as an `int`.
long	**longValue**() Returns the value of this `Integer` as a `long` after a widening primitive conversion.
static int	**max**(int a, int b) Returns the greater of two `int` values as if by calling **Math.max**.
static int	**min**(int a, int b) Returns the smaller of two `int` values as if by calling **Math.min**.
static int	**parseInt**(**String** s) Parses the string argument as a signed decimal integer.
short	**shortValue**() Returns the value of this `Integer` as a `short` after a narrowing primitive conversion.
static int	**sum**(int a, int b) Adds two integers together as per the + operator.
String	**toString**()

	Returns a `String` object representing this `Integer`'s value.
static `String`	**`toString`**`(int i)` Returns a `String` object representing the specified integer.
static `Integer`	**`valueOf`**`(int i)` Returns an `Integer` instance representing the specified `int` value.
static `Integer`	**`valueOf`**`(`**`String`**` s)` Returns an `Integer` object holding the value of the specified `String`.

Exercici 7.8: Crear una classe **Integer1** que simuli la classe Integer imitant tota la funcionalitat d'aquests mètodes.