Module 8
Multidimensional Arrays

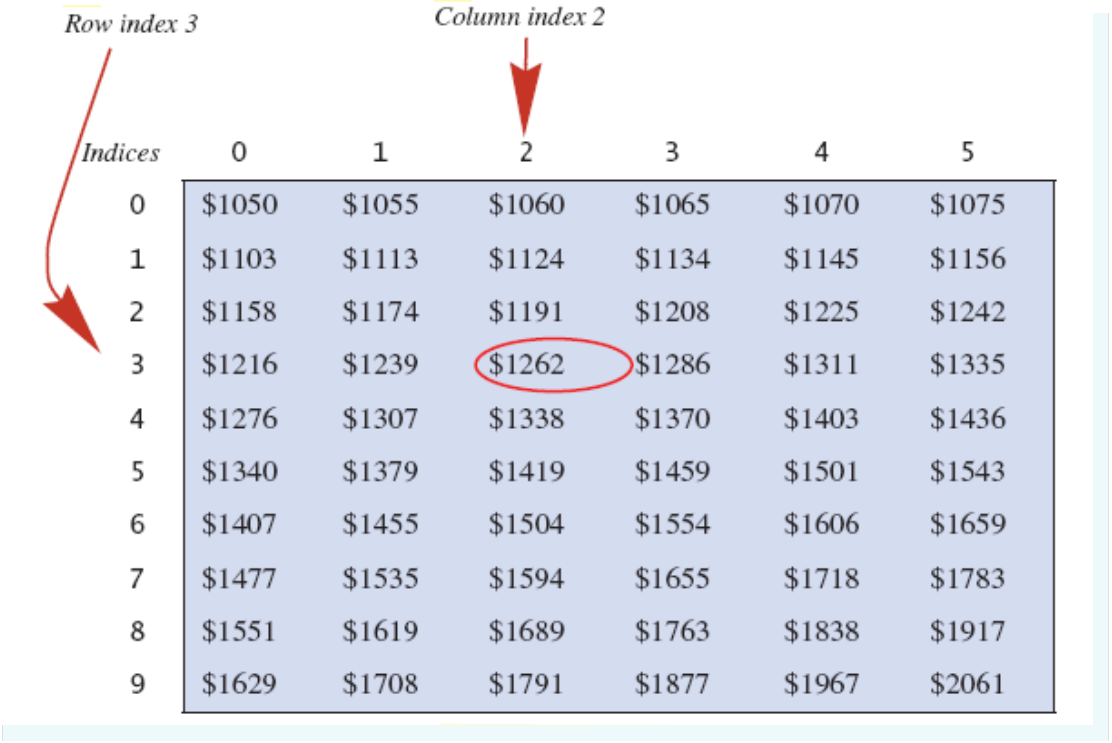
Multidimensional arrays

Multidimensional arrays are **data structures** that allow you to store data.

A multidimensional array is an array-of-arrays

Multidimensional arrays

We can use the row index (3) and the column index (2) to retrieve the desired value.



Indices	0	1	2	3	4	5
0	$1050	$1055	$1060	$1065	$1070	$1075
1	$1103	$1113	$1124	$1134	$1145	$1156
2	$1158	$1174	$1191	$1208	$1225	$1242
3	$1216	$1239	$1262	$1286	$1311	$1335
4	$1276	$1307	$1338	$1370	$1403	$1436
5	$1340	$1379	$1419	$1459	$1501	$1543
6	$1407	$1455	$1504	$1554	$1606	$1659
7	$1477	$1535	$1594	$1655	$1718	$1783
8	$1551	$1619	$1689	$1763	$1838	$1917
9	$1629	$1708	$1791	$1877	$1967	$2061

`table[3][2]` has a value of 1262

The first index refers to the row, and the second index to the column.

Multidimensional array declaration

```
//Method 1: Declaration and instantiation
int[][] x1 = new int[3][3];
int x2[][] = new int[3][3];

// Method 2: Explicit initialization
int[][] x3 = {{1,2,3},{4,5,6},{7,8,9}};

//Method 3: Declaration and instantiation
int[][] x4 = new int[3][];
x4[0] = new int[3];
x4[1] = new int[3];
x4[2] = new int[3];
```

General declaration

```
data_type[][] variable_name = new data_type[rows][columns];
```

data_type: String, char, Scanner, int

variable_name: Identificador mediante el cual nos vamos a referir a la variable

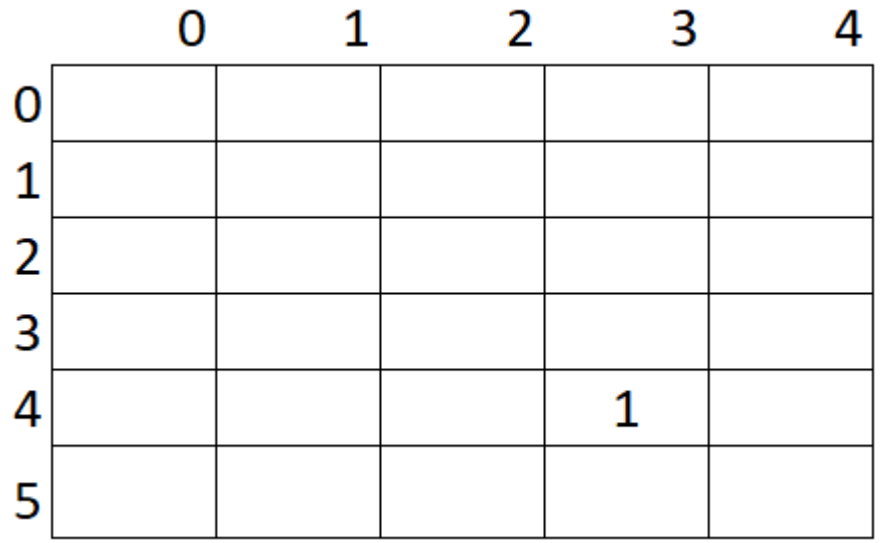
rows y columns: Cantidad de filas y columnas

Multidimensional arrays

	0	1	2	3	4
0	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
1	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
2	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
3	[3][0]	[3][1]	[3][2]	[3][3]	[3][4]
4	[4][0]	[4][1]	[4][2]	[4][3]	[4][4]
5	[5][0]	[5][1]	[5][2]	[5][3]	[5][4]

Array declaration

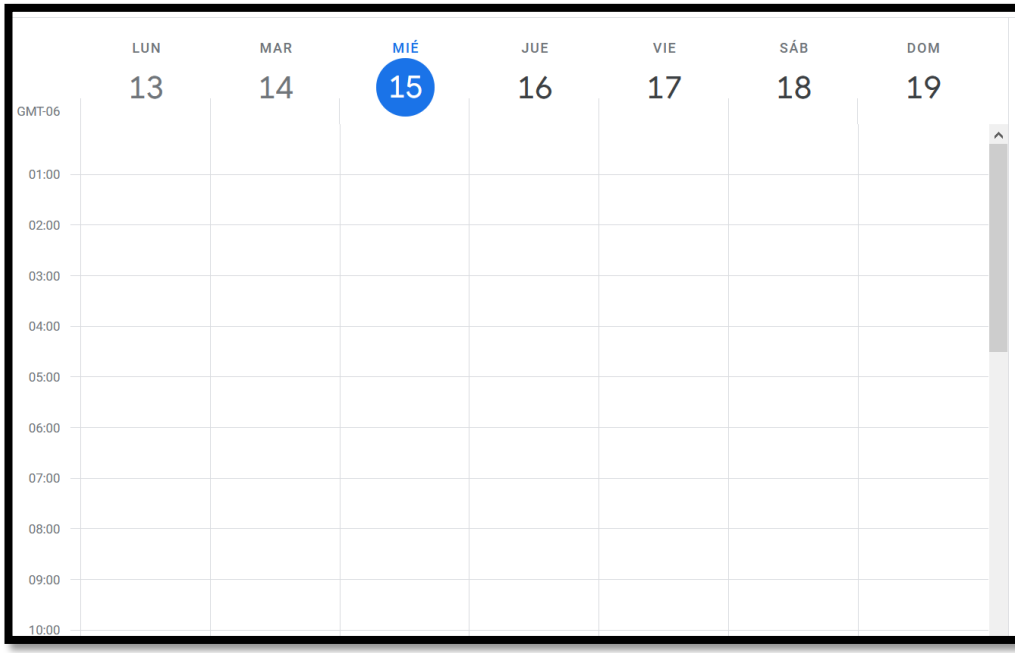
```
//matrix with:
// rows = 6
// columns = 5
int[][] matrix = new int[6][5];
matrix[4][3] = 1;
```



Example: Calendar

Design a method `createCalendar()` that instantiates and returns a multidimensional array that represents a calendar.

Each row will represent a 30-minute slot, and each column will represent one day of the week.




```
public static String[][] createCalendar(){
  int rows = 48;      // 48 slots of 30 minutes in a day
  int columns = 7;  // 7 days in a week

  String[][] calendar = new String[rows][columns];
  return calendar;
}
```

Matrix dimensions

The number of rows can be retrieved using the `length` attribute .

```
matrixName.length
```

The number of columns depends on each row, so we must use:

```
matrixName[row_index].length
```

Matrix

When we need to traverse a matrix, we use **nested loops**.

A nested loop means a loop inside of a loop.

```
for(int row = 0; row < matrix.length; row++){
  for(int column = 0; column < matrix[row].length; column++) {
    // Code here
  }
}
```

Initializing a matrix

We can use a cycle:

```
int[][] matrix = new int[3][3];
int counter = 1;

for(int row = 0; row < matrix.length; row++){
  for(int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] =  counter++;
  }
}
```

1	2	3
4	5	6
7	8	9

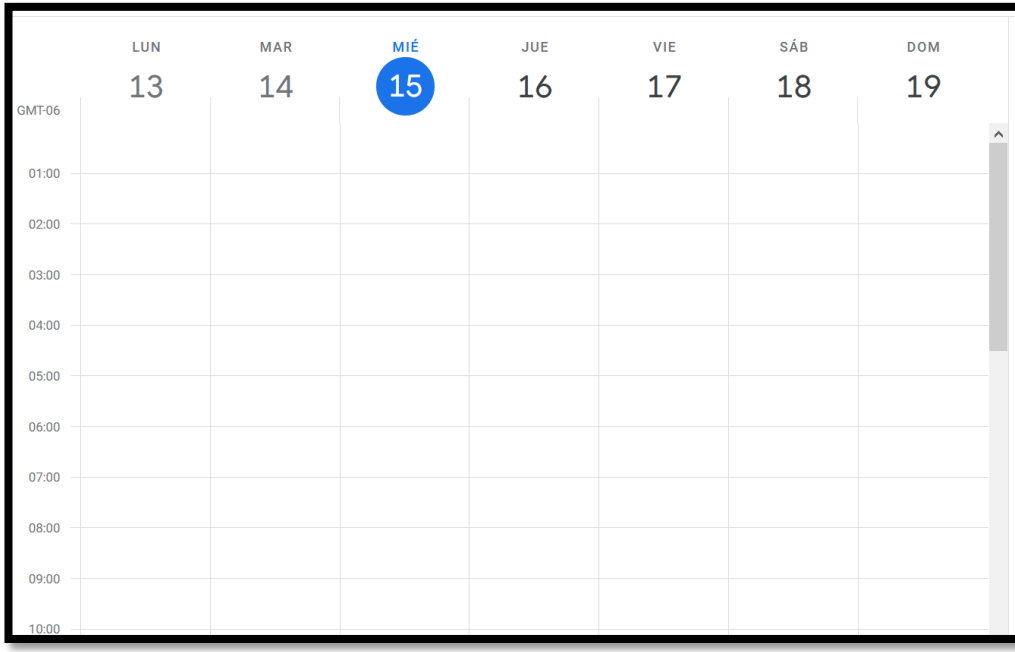
Add event

Design a method that adds an event to the calendar.

Inputs are:

- Calendar reference (as a String matrix)
- Name of the event (as a String)
- Day (as an integer)
- Time (as an integer)

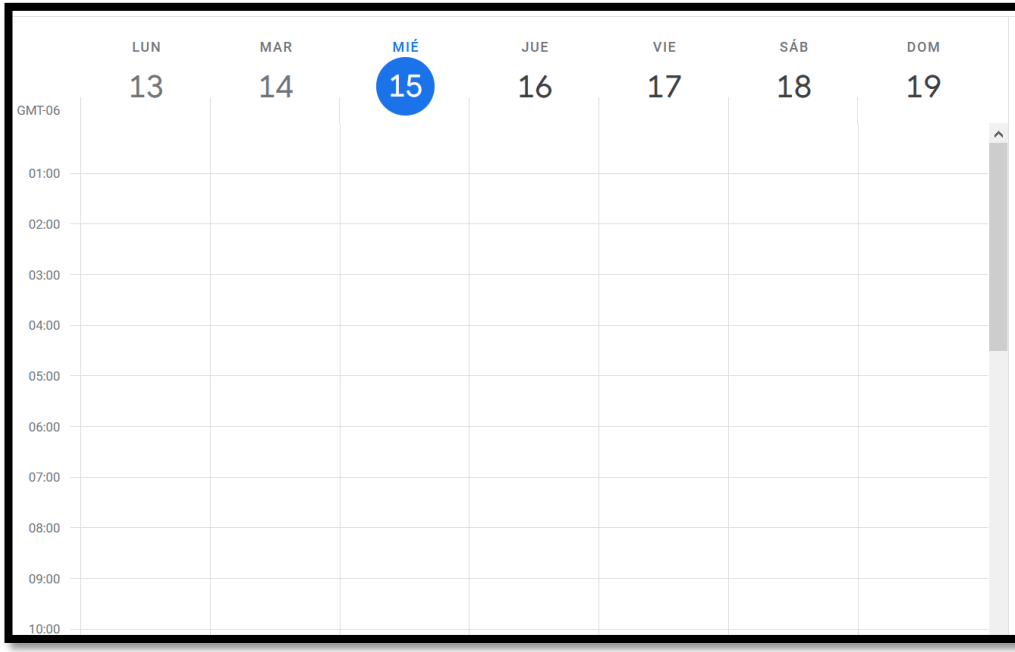
If the even can be added, put it on the calendar and return TRUE. Otherwise, return FALSE.




```
public static boolean addEvent(String[][] calendar, String event, int day, int time){
  // error!
  if (calendar == null || calendar.length <= time || calendar[time].length <= day) {
    return false;
  }
  if (calendar[time][day] != null)
     return false;

  calendar[time][day] =  event;
  return true;
}
```

Challenge!

Include a new parameter “duration” that allows the programmer to indicate how long the event will last. Assume the duration will be received in multiples of 30 minutes.



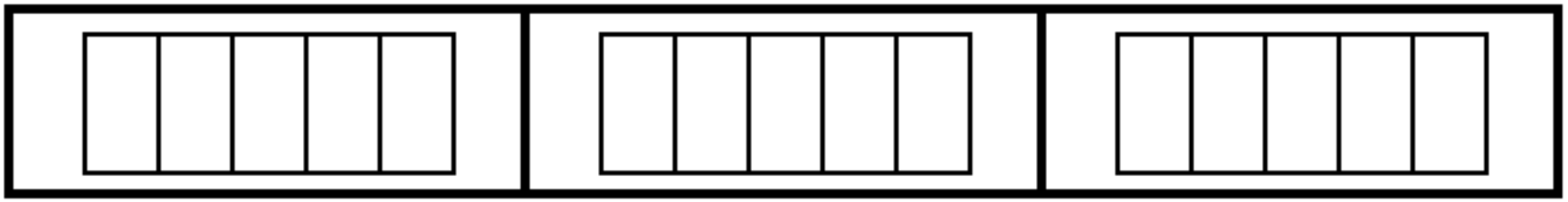

```
public static boolean addEvent(String[][] calendar, String event, int day, int
time, int duration){




}
```

Representing a Matrix

A matrix can be represented as a table, or an array of arrays.

Ejemplo:

```
int[][] table = new int[3][5];
```



Ragged Arrays

Not all elements of an array have to be of the same size.

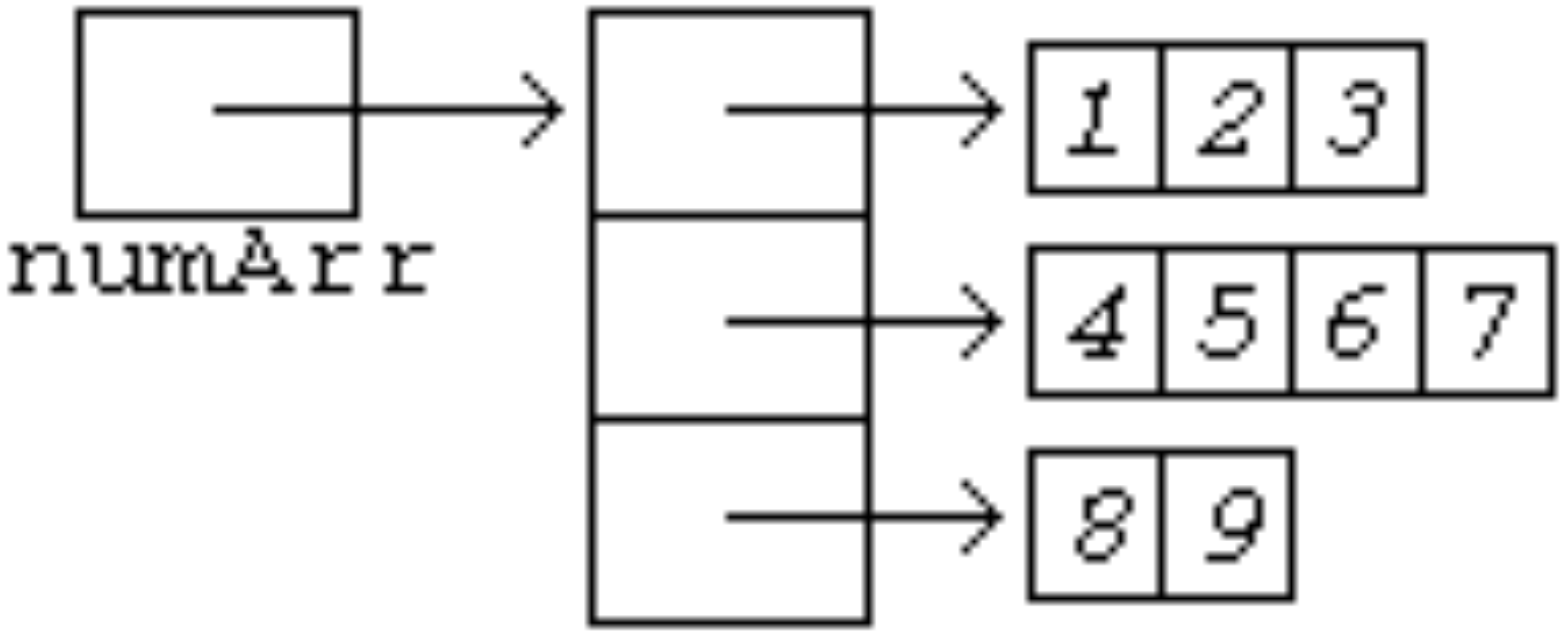
For example:

```
int[][] b;
b = new int[3][];
b[0] = new int[5]; //First row, 5 elements
b[1] = new int[7]; //Second row, 7 elements
b[2] = new int[4]; //Third row, 4 elements
```

Ragged arrays can be instantiated implicitly. Both arrays (a and b) are identical.

```
int[][] b;
b = new int[3][];
b[0] = new int[5]; //First row, 5 elements
b[1] = new int[7]; //Second row, 7 elements
b[2] = new int[4]; //Third row, 4 elements

int[][] c = {{0,0,0,0,0},       //First row, 5 elements
             {0,0,0,0,0,0,0}, //Second row, 7 elements
             {0,0,0,0}};       //Third row, 4 elements
```

Ragged Arrays



Layout of the same array

Example

Declare and initialize a matrix of integers to represent the month of May 2024. Notice that not all weeks have the same number of days.

Assign the number of the day to each element.

2024 MAY

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

www.GrabCalendar.com