Al-Mustaqbal University College (MUC)

Intelligent Medical Systems Department

Computer Programing II

Lecturer: Maytham N. Meqdad

Lecture 2

MULTIDIMENSIONAL ARRAYS

Example

int[ ][ ] myNumbers = { {1, 2, 3, 4}, {5, 6, 7؛{ {
System.out.println(myNumbers[1][2]); // Outputs 7

We can also use a `for loop` inside another `for loop` to get the elements of a two-dimensional array (we still have to point to the two indexes):

Example

```
public class Main {
  public static void main(String[] args) {
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    for (int i = 0; i < myNumbers.length; ++i) {
      for(int j = 0; j < myNumbers[i].length; ++j) {
        System.out.println(myNumbers[i][j]);
      }
    }
  }
}
```

Output: 1 2 3 4 5 6 7

Example of a multidimensional array in Java:

int[][] arr = new int[3][4];

arr[0][0] = 1;

arr[0][1] = 2;

arr[0][2] = 3;

arr[0][3] = 4;

arr[1][0] = 5;

arr[1][1] = 6;

arr[1][2] = 7;

arr[1][3] = 8;

arr[2][0] = 9;

arr[2][1] = 10;

```
arr[2][2] = 11;

arr[2][3] = 12;

// Print the elements of the array

for (int i = 0; i < arr.length; i++) {

    for (int j = 0; j < arr[i].length; j++) {

        System.out.print(arr[i][j] + " ");

    }

    System.out.println();

}
```

This creates a 2D array with 3 rows and 4 columns, and initializes each element with a value. The nested for loops are used to iterate over the elements of the array and print them out. The output would be:

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

Example of a 3D array in Java:

```
int[][][] arr = new int[3][4][2];
arr[0][0][0] = 1;
arr[0][0][1] = 2;
arr[0][1][0] = 3;
arr[0][1][1] = 4;
arr[0][2][0] = 5;
arr[0][2][1] = 6;
arr[0][3][0] = 7;
arr[0][3][1] = 8;
arr[1][0][0] = 9;
arr[1][0][1] = 10;
arr[1][1][0] = 11;
arr[1][1][1] = 12;
arr[1][2][0] = 13;
arr[1][2][1] = 14;
arr[1][3][0] = 15;
arr[1][3][1] = 16;
```

```
arr[2][0][0] = 17;
arr[2][0][1] = 18؛
arr[2][1][0] = 19؛
arr[2][1][1] = 20؛
arr[2][2][0] = 21؛
arr[2][2][1] = 22؛
arr[2][3][0] = 23؛
arr[2][3][1] = 24؛

 //Print the elements of the array
for (int i = 0; i < arr.length; i} (++
    for (int j = 0; j < arr[i].length; j} (++
        for (int k = 0; k < arr[i][j].length; k} (++
            System.out.print(arr[i][j][k؛(" " + [
{
        System.out.println؛()
{
    System.out.println؛()
{
```

This creates a 3D array with 3 "layers", each with 4 rows and 2 columns, and initializes each element with a value. The nested for loops are used to iterate over the elements of the array and print them out. The output would be:

```
1 2
3 4
5 6
7 8

9 10
11 12
13 14
15 16

17 18
19 20
21 22
23 24
```

Example of a 3D array in Java that stores names:

```
String[ ][ ][ ] names = {
    {{"Maytham", "Ali"}, {"Hassan", "Abbas"}},
    {{"Nabeel", "Ahmed"}, {"Sara", "Fatima"}},
    {{"Meqdad", "Ammar"}, {"Zainab", "Maryam"}}
};

// Print the elements of the array
for (int i = 0; i < names.length; i++) {
    for (int j = 0; j < names[i].length; j++) {
        for (int k = 0; k < names[i][j].length; k++) {
            System.out.print(names[i][j][k] + " ");
        }
        System.out.println();
    }
    System.out.println();
}
```

This creates a 3D array with 3 "layers", each with 2 rows and 2 columns, and initializes each element with a name. The nested for loops are used to iterate over the elements of the array and print them out. The output would be:

```
Maytham Ali
Hassan Abbas

Nabeel Ahmed
Sara Fatima

Meqdad Ammar
Zainab Maryam
```

Example :::Here is the code to print a multiplication table of 2 using a multidimensional array in Java:

```
public class MultiplicationTable {
    public static void main(String[] args) {
        int[][] table = new int[10][10];

        // Populate the array with multiplication table values
        for (int i = 0; i < table.length; i++) {
            for (int j = 0; j < table[i].length; j++) {
                table[i][j] = 2 * (i+1) * (j+1);
            }
        }
```

```
        // Print the multiplication table
        for (int i = 0; i < table.length; i++) {
            for (int j = 0; j < table[i].length; j++) {
                System.out.print(table[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

In this code, we have initialized a 10x10 multidimensional array called table. We then use two nested loops to populate the array with multiplication table values, where each element of the array is equal to 2 times the product of its row and column indices. Finally, we use another set of nested loops to print the multiplication table to the console. The output will be a multiplication table of 2

```
4 6 8 10 12 14 16 18 20 22
6 12 18 24 30 36 42 48 54 60
8 18 24 32 40 48 56 64 72 80
10 24 32 40 50 60 70 80 90 100
12 30 40 50 60 72 84 96 108 120
14 36 48 60 72 84 98 112 126 140
16 42 56 70 84 98 112 128 144 160
18 48 64 80 96 112 128 144 162 180
20 54 72 90 108 126 144 162 180 200
22 60 80 100 120 140 160 180 200 220
```

==========

An array having more than one index is sometimes useful. For example, suppose you wanted to store the dollar amounts shown in Figure 7.6 in some sort of array. The highlighted part of the table contains 60 such entries. If you use an array that has one index, the length of the array would be 60. Keeping track of which entry goes with which index number would be difficult. On the other hand, if you allow yourself two indices, you can use one index for the row of this table and one index for the column. Such an array is illustrated in Figure 7.7.

Arrays that have exactly two indices can be displayed on paper as a two-dimensional table and are called **two-dimensional arrays**. By convention, we think of the first index as denoting the row of the table and the second index as denoting the column. Note that, as was true for the simple arrays we have already seen, we begin numbering indices with 0 rather than 1. The Java notation for an element of a two-dimensional array is

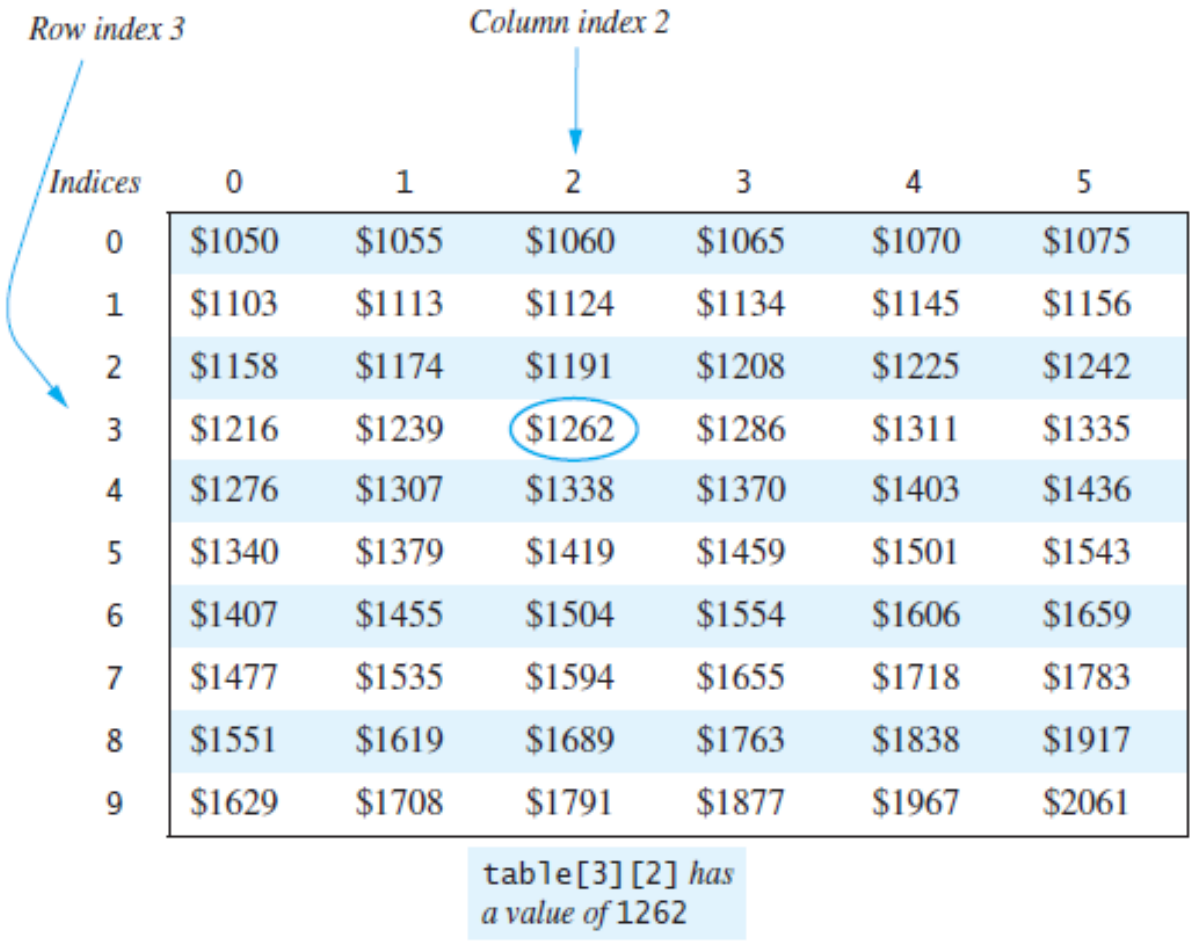
```
Array_Name[Row_Index][Column_Index]
```

For example, if the array is named `table` and it has two indices, `table[3][2]` is the entry whose row index is 3 and column index is 2. Because indices begin

Savings Account Balances for Various Interest Rates Compounded Annually (Rounded to Whole Dollar Amounts)

Year	5.00%	5.50%	6.00%	6.50%	7.00%	7.50%
1	$1050	$1055	$1060	$1065	$1070	$1075
2	$1103	$1113	$1124	$1134	$1145	$1156
3	$1158	$1174	$1191	$1208	$1225	$1242
4	$1216	$1239	$1262	$1286	$1311	$1335
5	$1276	$1307	$1338	$1370	$1403	$1436
6	$1340	$1379	$1419	$1459	$1501	$1543
7	$1407	$1455	$1504	$1554	$1606	$1659
8	$1477	$1535	$1594	$1655	$1718	$1783
9	$1551	$1619	$1689	$1763	$1838	$1917
10	$1629	$1708	$1791	$1877	$1967	$2061

FIGURE 7.7 Row and Column Indices for an Array Named `table`



at 0, this entry is in the fourth row and third column of `table`, as Figure 7.7 illustrates. Trying to relate array indices to actual row and column numbers gets confusing and is likely to be unnecessary.

n-dimensional array

Arrays having more than one index are generally called **multidimensional arrays**. More specifically, an array that has n indices is said to be an **n-dimensional array**. Thus, an ordinary one-index array is a **one-dimensional array**. Although arrays having more than two dimensions are rare, they can be useful for some applications.

Multidimensional-Array Basics

Arrays having multiple indices are handled in much the same way as one-dimensional arrays. To illustrate the details, we will take you through an example Java program that displays an array like the one in Figure 7.7. The program is shown in Listing 7.12. The array is called `table`. The following statement declares the name `table` and creates the array:

Declaring and creating a two-dimensional array

```
int[][] table = new int[10][6];
```

```
/**
 Displays a two-dimensional table showing how
 interest rates affect bank balances.
*/
public class InterestTable
{
    public static void main(String[] args)
    {
        int[][] table = new int[10][6];
        for (int row = 0; row < 10; row++)
            for (int column = 0; column < 6; column++)
                table[row][column] =
                    getBalance(1000.00, row + 1, (5 + 0.5 *
                                                  column));
        System.out.println("Balances for Various Interest Rates " +
                           "Compounded Annually");
        System.out.println("(Rounded to Whole Dollar Amounts)");
        System.out.println();
        System.out.println("Years 5.00% 5.50% 6.00% 6.50% " +
                           "7.00% 7.50%");
        for (int row = 0; row < 10; row++)
        {
            System.out.print((row + 1) + " ");
            for (int column = 0; column < 6; column++)
                System.out.print("$" + table[row][column] + " ");
            System.out.println();
        }
    }

    /**
    Returns the balance in an account after a given number of years
    and interest rate with an initial balance of startBalance.
    Interest is compounded annually. The balance is rounded
    to a whole number.
    */
    public static int getBalance(double startBalance, int years,
                                 double rate)
    {
        double runningBalance = startBalance;
        for (int count = 1; count <= years; count++)
            runningBalance = runningBalance * (1 + rate / 100);
        return (int)(Math.round(runningBalance));
    }
}
```

A real application would do something more with the array `table`. *This is just a demonstration program.*

(continued)

Sample Screen Output

```
Balances for Various Interest Rates Compounded Annually
(Rounded to Whole Dollar Amounts)

Years    5.00%    5.50%    6.00%    6.50%    7.00%    7.50%
1        $1050    $1055    $1060    $1065    $1070    $1075
2        $1103    $1113    $1124    $1134    $1145    $1156
3        $1158    $1174    $1191    $1208    $1225    $1242
4        $1216    $1239    $1262    $1286    $1311    $1335
5        $1276    $1307    $1338    $1370    $1403    $1436
6        $1340    $1379    $1419    $1459    $1501    $1543
7        $1407    $1455    $1504    $1554    $1606    $1659
8        $1477    $1535    $1594    $1655    $1718    $1783
9        $1551    $1619    $1689    $1763    $1838    $1917
10        $1629    $1708    $1791    $1877    $1967    $2061
```

The last line is out of alignment because 10 has two digits. This is easy to fix, but that would clutter the discussion of arrays with extraneous concerns.

RECAP Declaring and Creating a Multidimensional Array

You declare a name for a multidimensional array and then create the array in basically the same way that you declare and create a one-dimensional array. You simply use as many square brackets as there are indices.

SYNTAX

```
Base_Type[]...[]Array_Name = new Base_Type[Length_1]...
                             [Length_n];
```

EXAMPLES

```
char[][] page = new char[100][80];
int[][] table = new int[10][6];
double[][][]threeDPicture = new double[10][20][30];
SomeClass[][]entry = new SomeClass[100][80];
```

`SomeClass` is a class.

PROGRAMMING EXAMPLE Employee Time Records

In this programming example, a two-dimensional array named `hours` is used to store the number of hours worked by each employee of a company for each of the five days Monday through Friday. The first array index is used to designate a day of the week, and the second array index is used to designate an employee. The two-dimensional array is a private instance variable in the class named `TimeBook` given in Listing 7.14. The class includes a method `main` that demonstrates the class for a small company having only three employees. The employees are numbered 1, 2, and 3 but are stored at array index positions 0, 1, and 2, since the array indices are numbered starting with 0. Thus, an adjustment of minus 1 is sometimes needed when specifying an employee's array index. We can number days as 0 for Monday, 1 for Tuesday, and so forth to avoid adjustments between day numbers and day indices.

For example, the hours worked by employee number 3 on Tuesday are recorded in `hours[1][2]`. The first index denotes the second workday of the week—Tuesday—and the second index denotes the third employee.

The class `TimeBook` shown in Listing 7.14 is not yet complete. It needs more methods to be a really useful class, but it has enough methods for the demonstration program in `main`. You can think of the definition in Listing 7.14 as a first pass at writing the class definition. It even has a stub for the definition of the method `setHours`. Recall that a stub is a method whose definition can be used for testing but is not yet the final definition. At this stage, however, `setHours` is complete enough to illustrate the use of the two-dimensional array `hours`, which is an instance variable of the class.

`setHours` is a stub

In addition to the two-dimensional array `hours`, the class `TimeBook` uses two one-dimensional arrays as instance variables: The array `weekHours` records the total hours each employee works in a week. That is, `weekHours[0]` is the total number of hours worked by employee 1 in the week, `weekHours[1]` is the total number of hours worked by employee 2 in the week, and so forth. The array `dayHours` records the total number of hours worked by all the employees on each day of the week. That is, `dayHours[MON]` is the total number of hours worked on Monday by all of the employees combined, `dayHours[TUE]` is the total number of hours worked on Tuesday by all of the employees, and so on.

LISTING 7.14 A Timekeeping Program *(part 1 of 4)*

```
/**
 Class that records the time worked by each of a
 company's employees during one five-day week.
 A sample application is in the main method.
*/
public class TimeBook
{
    private int numberOfEmployees;
    private int[][] hours;    //hours[i][j] has the hours for
                              //employee j on day i.
    private int[] weekHours; //weekHours[i] has the week's
                              //hours worked for employee i + 1.
    private int[] dayHours;  //dayHours[i] has the total hours
                              //worked by all employees on day i.
    private static final int NUMBER_OF_WORKDAYS = 5;
    private static final int MON = 0;
    private static final int TUE = 1;
    private static final int WED = 2;
    private static final int THU = 3;
    private static final int FRI = 4;
```

```
/**
 Reads hours worked for each employee on each day of the
 work week into the two-dimensional array hours. (The method
 for input is just a stub in this preliminary version.)
 Computes the total weekly hours for each employee and
 the total daily hours for all employees combined.
*/
public static void main(String[] args)
{
    private static final int NUMBER_OF_EMPLOYEES = 3;
    TimeBook book = new TimeBook(NUMBER_OF_EMPLOYEES);
    book.setHours();
    book.update();
    book.showTable();
}
```

A class generally has more methods. We have defined only the methods used in `main`.

```
public TimeBook(int theNumberOfEmployees)
{
    numberOfEmployees = theNumberOfEmployees;
    hours = new int[NUMBER_OF_WORKDAYS][numberOfEmployees];
    weekHours = new int[numberOfEmployees];
    dayHours = new int[NUMBER_OF_WORKDAYS];
}
```

The **final** *program would obtain the employee data from the user.*

```
public void setHours() //This is a stub.
{
    hours[0][0] = 8; hours[0][1] = 0; hours[0][2] = 9;
    hours[1][0] = 8; hours[1][1] = 0; hours[1][2] = 9;
    hours[2][0] = 8; hours[2][1] = 8; hours[2][2] = 8;
    hours[3][0] = 8; hours[3][1] = 8; hours[3][2] = 4;
    hours[4][0] = 8; hours[4][1] = 8; hours[4][2] = 8;
}

public void update()
{
    computeWeekHours();
    computeDayHours();
}

private void computeWeekHours()
{
    for (employeeNumber = 1; employeeNumber <=
         numberOfEmployees; employeeNumber++)
```

(continued)

```
        {//Process one employee:
           int sum = 0;
           for (int day = MON; day <= FRI; day++)
               sum = sum + hours[day][employeeNumber - 1];
               //sum contains the sum of all the hours worked in
               //one
               //week by the employee with number employeeNumber.
           weekHours[employeeNumber - 1] = sum;
        }
    }

    private void computeDayHours()
    {
        for (int day = MON; day <= FRI; day++)
        {//Process one day (for all employees):
           int sum = 0;
           for (int employeeNumber = 1;
                    employeeNumber <= numberOfEmployees;
                    employeeNumber++)
               sum = sum + hours[day][employeeNumber - 1];
               //sum contains the sum of all hours worked by all
               //employees on one day.
           dayHours[day] = sum;
        }
    }
```

The method showTable can and should be made more robust Programming Project 6.

```
public void showTable()
{
    // heading
    System.out.print("Employee ");
    for (int employeeNumber = 1;
             employeeNumber <= numberOfEmployees;
             employeeNumber++)
        System.out.print(employeeNumber + " ");
    System.out.println("Totals");
    System.out.println( );

    // row entries
    for (int day = MON; day <= FRI; day++)
    {
        System.out.print(getDayName(day) + " ");
        for (int column = 0; column < hours[day].length;
                 column++)
            System.out.print(hours[day][column] + " ");
        System.out.println(dayHours[day]);
    }
```

```
        System.out.println( );
        System.out.print("Total = ");
        for (int column = 0; column < numberOfEmployees; column++)
            System.out.print(weekHours[column] + " ");
        System.out.println( );
    }
    //Converts 0 to "Monday", 1 to "Tuesday", etc.
    //Blanks are inserted to make all strings the same length.
    private String getDayName(int day)
    {
        String dayName = null;
        switch (day)
        {
            case MON:
                dayName = "Monday ";
                break;
            case TUE:
                dayName = "Tuesday ";
                break;
            case WED:
                dayName = "Wednesday";
                break;
            case THU:
                dayName = "Thursday ";
                break;
            case FRI:
                dayName = "Friday ";
                break;
            default:
                System.out.println("Fatal Error.");
                System.exit(0);
                break;
        }
        return dayName;
    }
}
```

Sample Screen Output

```
Employee    1   2   3   Totals
Monday      8   0   9   17
Tuesday     8   0   9   17
Wednesday   8   8   8   24
Thursday    8   8   4   20
Friday      8   8   8   24
Total  =    40  24  38
```

FIGURE 7.8 Arrays for the Class `TimeBook`