7.5 Multidimensional Arrays

- Arrays with more than one index
  - » number of dimensions = number of indexes
- Arrays with more than two dimensions are a simple extension of two-dimensional (2-D) arrays
- A 2-D array corresponds to a table or grid
  - » one dimension is the row
  - » the other dimension is the column
  - » Cell : an intersection of a row and column
  - » an array element corresponds to a cell in the table

Table as a 2-Dimensional Array

- The table assumes a starting balance of $1000
- First dimension: row identifier - Year
- Second dimension: column identifier - percentage
- Cell contains balance for the year (row) and percentage (column)
- Balance for year 4, rate 7.00% = $1311

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
…	…	…	…	…	…	…

Table as a 2-D Array

Column Index 4
(5th column)

Row Index 3
(4th row)

Indexes	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
…	…	…	…	…	…	…

- Generalizing to two indexes: [row][column]
- First dimension: row index
- Second dimension: column index
- Cell contains balance for the year/row and percentage/column
- All indexes use zero-numbering
  - » Balance[*3*][*4*] = cell in 4th row (year = 4) and 5th column (7.50%)
  - » Balance[3][4] = $1311 (shown in yellow)

Java Code to Create a 2-D Array

- Syntax for 2-D arrays is similar to 1-D arrays

- Declare a 2-D array of `int`s named `table`
  - » the table should have ten rows and six columns

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

Processing a 2-D Array: `for` Loops Nested 2-Deep

- Arrays and `for` loops are a natural fit
- To process all elements of an *n*-D array nest *n* for loops
  - » each loop has its own counter that corresponds to an index
- For example: calculate and enter balances in the interest table
  - » **inner loop** repeats 6 times (six rates) for every outer loop iteration
  - » the outer loop repeats 10 times (10 different values of `years`)
  - » so the inner repeats 10 x 6 = 60 times = # cells in `table`

```
int[][] table = new int[10][6];
int row, column;
for (row = 0; row < 10; row++)
    for (column = 0; column < 6; column++)
        table[row][column] = balance(1000.00, row + 1, (5 + 0.5*column));
```

Excerpt from `main` method of `InterestTable`

Listing 7.12 Using a Two-Dimensional Array - InterestTable.java

```
// Listing 7.12 Using a Two-Dimensional Array

/**
 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];
      int row, column;
      for (row = 0; row < 10; row++)
         for (column = 0; column < 6; column++)
            table[row][column] =
               balance(1000.00, row + 1, (5 + 0.5*column));
```

```
      System.out.println("Balances for Various Interest Rates");
      System.out.println("Compounded Annually");
      System.out.println("(Rounded to Whole Dollar Amounts)");
      System.out.println("Years  5.00%  5.50%  6.00%  6.50%  7.00%  7.50%");
      System.out.println( );
      for (row = 0; row < 10; row++)
      {
        System.out.print((row + 1) + "      ");
          for (column = 0; column < 6; column++)
            System.out.print("$" + table[row][column] + "  ");
        System.out.println( );
      }
   }
   /**
    Returns the balance in an account that starts with startBalance
    and is left for the indicated number of years with rate as the
    interest rate. Interest is compounded annually. The balance is
    rounded to a whole number.
   */
   public static int balance(double startBalance, int years, double rate)
   {
      double runningBalance = startBalance;
      int count;
      for (count = 1; count <= years; count++)
         runningBalance = runningBalance*(1 + rate/100);
      return (int) (Math.round(runningBalance));
   }
}
```

Method to Calculate the Cell Values

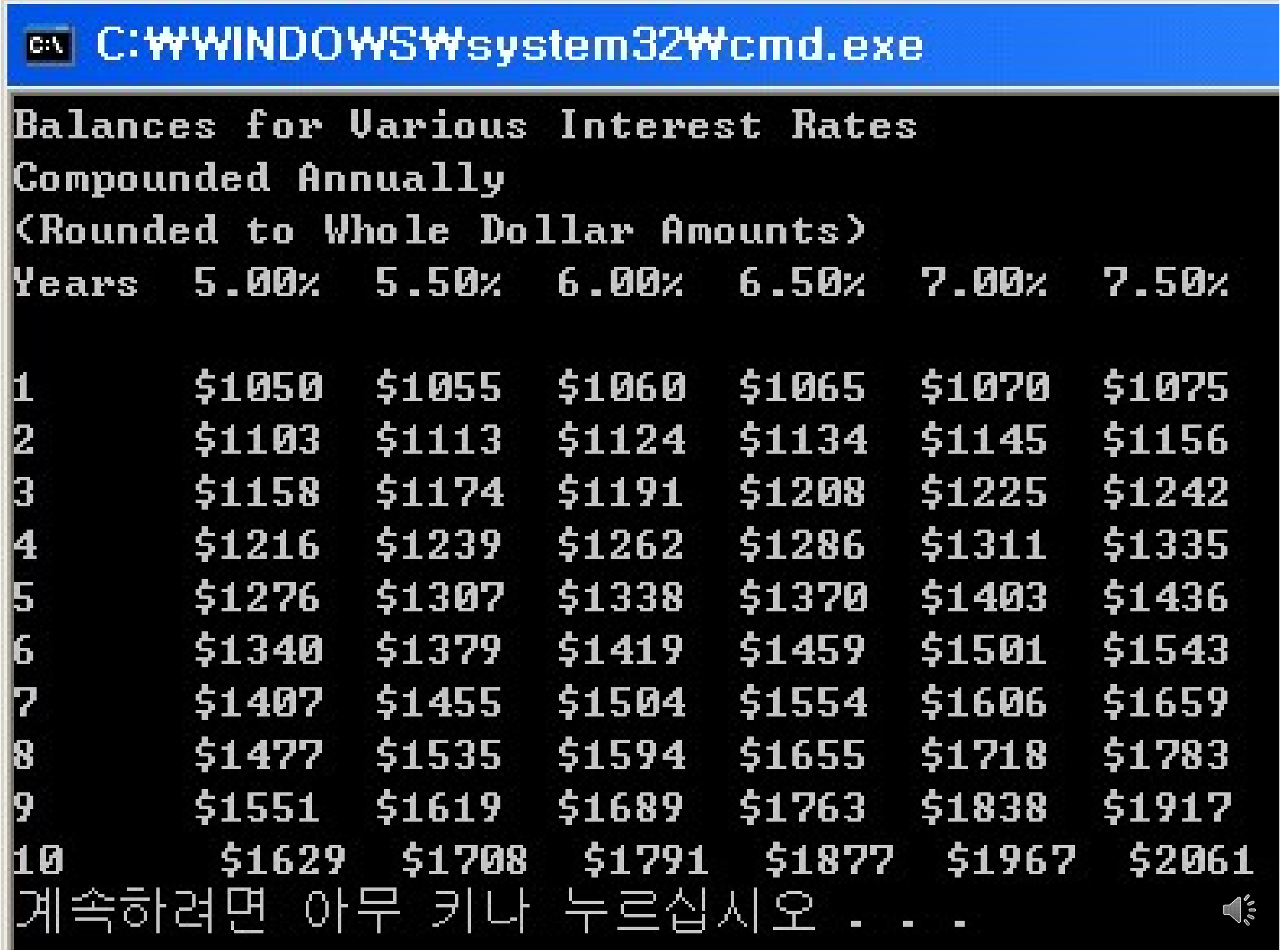
Each array element corresponds to the balance for a specific number of years and a specific interest rate (assuming a starting balance of $1000):

balance(starting, years, rate) = (starting) x $(1 + \text{rate})^{\text{years}}$

The repeated multiplication by (1 + rate) can be done in a `for` loop that repeats `years` times.

```
public static int balance(double startBalance, int years, double rate)
 {
    double runningBalance = startBalance;
    int count;
    for (count = 1; count <= years; count++)
       runningBalance = runningBalance*(1 + rate/100);
    return (int) (Math.round(runningBalance));
 }
```

`balance` method in class InterestTable


C:₩WINDOWS₩system32₩cmd.exe
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
계속하려면 아무 키나 누르십시오 . . .

Multidimensional Array Parameters and Returned Values

- Methods may have **multi-D array parameters**
- Methods may **return a multi-D array** as the value returned
- The situation is similar to 1-D arrays, but **with more brackets**
- Example: a 2-D `int` array as a method argument

```
public static void showTable(int[][] displayArray)
{
   int row, column;
   for (row = 0; row <                          ; row++)
   {
      System.out.print((row + 1) + "      ");
      for (column = 0; column <                          ; column++)
         System.out.print("$" + displayArray[row][column] + "  ");
      System.out.println();
   }
}
```

Notice how the number of rows is obtained

Notice how the number of columns is obtained

`showTable` method from class `InterestTable2`

Returning an array

```
public static                corner(double[][] startArray, int size)
  {
    double[][] temp = new double[size][size];
    int row, column;
    for (row = 0; row < size; row++)
       for (column = 0; column < size; column++)
          temp[row][column] = startArray[row][column];
       return temp;
  }
```

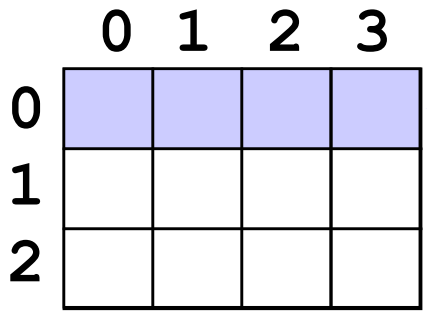
Implementation of Multidimensional Arrays

- Multidimensional arrays are implemented as ***arrays of arrays***.

  Example:

  ```
  int[][] table = new int[3][4];
  ```

  - » `table` is a one-dimensional array of length 3
  - » Each element in `table` is an array with base type `int`.
- Access a row by only using only one subscript:
  - » `table[0].length` gives the length (4) of the first row in the array



`table[0]` refers to the first row in the array, which is a one-dimensional array.

Note: `table.length` (which in this case) is not the same thing as `table[0].length` (which is

Listing 7.13

```
/**
Displays a two-dimensional table showing how interest
rates affect bank balances.
*/
public class InterestTable2
{
    public static final int ROWS = 10;
    public static final int COLUMNS = 6;
    public static void main (String [] args)
    {
        int [] [] table = new int [ROWS] [COLUMNS];
        for (int row = 0 ; row < ROWS ; row++)
            for (int column = 0 ; column < COLUMNS ; 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%");
     showTable (table);

  } //end main

public static void showTable (int [] [] anArray)
{
    for (int row = 0 ; row < ROWS ; row++)
    {
        System.out.print ((row + 1) + " ");
        for (int column = 0 ; column < COLUMNS ; column++)
            System.out.print ("$" + anArray [row] [column] + " ");
        System.out.println ();
    }
}
```

```
    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));
    }
}
```


C:₩Windows₩system32₩cmd.exe
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
계속하려면 아무 키나 누르십시오 . . .

Listing *.* The Method showTable Redefined. - InterestTable3.java

```
/**
 The array displayArray can have any dimensions.
 Postcondition: The array contents are displayed with dollar signs.
*/
public static void showTable(int[][] displayArray)
{
    int row, column;
    for (row = 0; row < displayArray.length; row++)
    {
      System.out.print((row + 1) + "      ");
      for (column = 0; column < displayArray[row].length; column++)
        System.out.print("$" + displayArray[row][column] + "  ");
      System.out.println( );
    }
 }
```

```
/**
   Precondition: The array displayArray has 10 rows and 6 columns.
   Postcondition: The array contents are displayed with dollar signs.
 */
 public static void showTable(int[][] displayArray)  //
 {
   int row, column;
   for (row = 0; row < 10; row++)
   {
      System.out.print((row + 1) + "      ");
      for (column = 0; column < 6; column++)
         System.out.print("$" + displayArray[row][column] + "  ");
      System.out.println( );
   }
 }
```

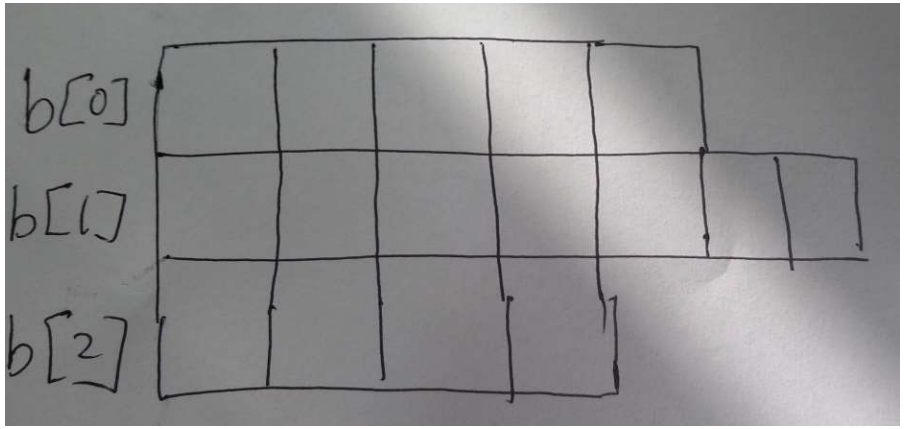
Ragged Arrays

- Ragged arrays have rows of [redacted] length
  - » each row has a different number of columns, or entries

- Ragged arrays are allowed in Java

- Example: create a 2-D `int` array named `b` with 5 elements in the first row, 7 in the second row, and 4 in the third row:

```
int[][] b;
b = new int[3][];
b[0] = new int[5];
b[1] = new int[7];
b[2] = new int[4];
```



Programming Example: Employee Time Records

Display 6.20. Time Keeping Program (TimeBook.java)

- The class `TimeBook` uses several arrays to keep track of employee time records:

```
public class TimeBook
{
        private int numberOfEmployees;
        private int[][] hours;
        private int[] weekHours;
        private int[] dayHours;
        . . .
}
```

`hours[i][j]` has the hours for employee `j` on day `i`

`weekHours[i]` has the week's hours for employee `i+1`

`dayHours[i]` has the total hours worked by all employees on day `i`