Using Two-Dimensional Arrays

Great news! What used to be the old one-floor Java Motel has just been renovated! The new, five-floor Java Hotel features a free continental breakfast and, at absolutely no charge, a free newspaper delivered to your door every morning. That's a 50-cent value, absolutely free!

Speaking of things that are continental, the designers of the new Java Hotel took care to number floors the way people do in France. The ground floor (in French, "le rez-de-chaussée") is the zero floor, the floor above that is the first floor, and so on. Figure B-1 shows the newly renovated hotel.

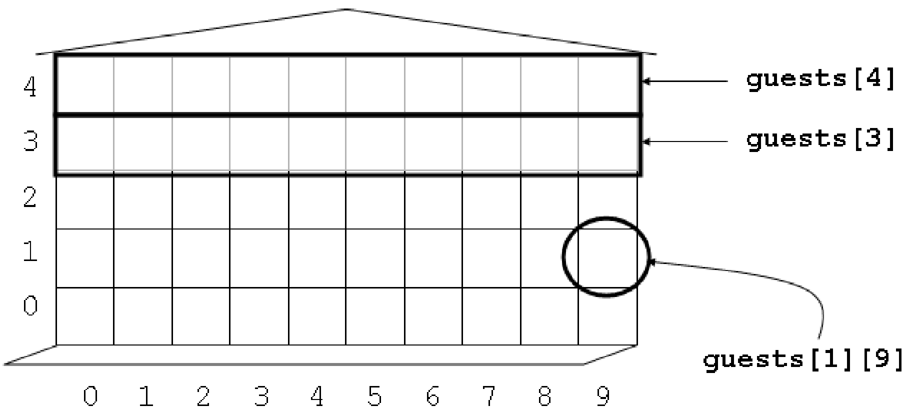


Figure B-1: A big, high-rise hotel.

A two-dimensional array of primitive values

You can think of the hotel as an array with two indices -- a *two-dimensional array*. You declare the array this way.

```
int guests[][] = new int[5][10];
```

The guests array has five rows (numbered 0 to 4, inclusive) and ten columns (numbered 0 to 9, inclusive). To register two guests in Room 9 on the first floor, you write

```
guests[1][9] = 2;
```

TechnicalStuff

The people who do serious Java like to think of a two-dimensional array as an array of rows (that is, an array of ordinary one-dimensional arrays). With this thinking, the rows of the guests array (above) are denoted guests[0], guests[1], guests[2], guests[3], and guests[4]. For a picture of all this, refer to Figure B-1.

A complete program that uses this guest array is shown in Listing B-1.

Listing B-1 An array of arrays

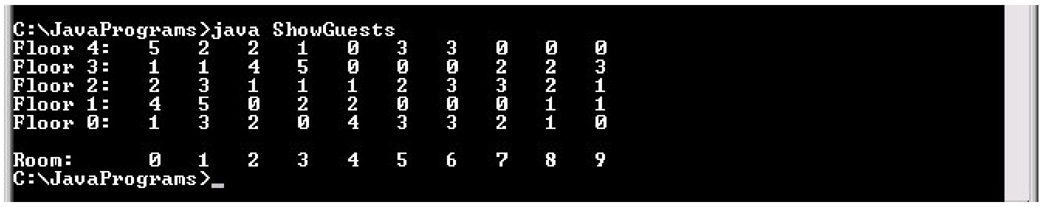
```
import static java.lang.System.out;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ShowGuests {

    public static void main(String args[]) throws
           FileNotFoundException {
        int guests[][] = new int[5][10];
        Scanner myScanner = new Scanner(new
           File("GuestList"));

        for (int floor = 0; floor < 5; floor++) {
            for (int roomNum = 0; roomNum < 10; roomNum++) {
                guests[floor][roomNum] = myScanner.nextInt();
            }
        }

        for (int floor = 4; floor >= 0; floor--) {
            out.print("Floor " + floor + ":");
            for (int roomNum = 0; roomNum < 10; roomNum++) {
                out.print("   ");
                out.print(guests[floor][roomNum]);
            }
            out.println();
        }

        out.println();
        out.print("Room:   ");
        for (int roomNum = 0; roomNum < 10; roomNum++) {
            out.print("   ");
```

```
                    out.print(roomNum);
                }
            }
        }
```

Figure B-2 shows a run of the code from Listing B-1. The input file, GuestList, looks like the file in Listing 11-1, except that the file for this section's program has 50 lines in it.

OnTheWeb

You can snare a 50-line GuestList file along with this document's code listings from the book's Web site.



```
C:\JavaPrograms>java ShowGuests
Floor 4:    5    2    2    1    0    3    3    0    0    0
Floor 3:    1    1    4    5    0    0    0    2    2    3
Floor 2:    2    3    1    1    1    2    3    3    2    1
Floor 1:    4    5    0    2    2    0    0    0    1    1
Floor 0:    1    3    2    0    4    3    3    2    1    0

Room:       0    1    2    3    4    5    6    7    8    9
C:\JavaPrograms>_
```

Figure B-2: Guest counts.

In Listing B-1, notice the primary way you handle a two-dimensional array -- by putting a for loop inside another for loop. For instance, when you read values into the array, you have a room number loop within a floor number loop.

```
for (int floor = 0; floor < 5; floor++) {
    for (int roomNum = 0; roomNum < 10; roomNum++) {
```

Because the roomNum loop is inside the floor loop, the roomNum variable changes faster than the floor variable. In other words, the program prints guest counts for all the rooms on a floor before marching on to the next floor.

Remember

The outer loop's variable changes slower; the inner loop's variable changes faster.

In displaying the hotel's numbers, I could have chosen to start with floor 0 and go up to floor 4. But then the output would have looked like an upside-down hotel. In the program's output, you want the top floor's numbers to be displayed first. To make this work, I created a loop whose counter goes backwards.

```
for (int floor = 4; floor >= 0; floor--)
```

So notice that the loop's counter starts at 4, goes downward each step of the way, and keeps going down until the counter's value is equal to 0.

A two-dimensional array of objects

This section does "one better" on the stuff from earlier sections. If you can make a two-dimensional array and an array of objects, then why not join these ideas to make a two-dimensional array of objects. Technically, this ends up being an array of arrays of objects. How about that!

First you define your two-dimensional array of Room objects. (The declaration of the Room class comes right from Listing 11-5.)

```
Room rooms[][] = new Room[5][10];
```

Next, you do that all-important step of constructing an object for each component in the array.

```
for (int floor = 0; floor < 5; floor++) {
    for (int roomNum = 0; roomNum < 10; roomNum++) {
        rooms[floor][roomNum] = new Room();
```

Then you read values into the array components' variables, write values, and so on. A complete program is shown in Listing B-2.

Listing B-2 A two-dimensional array of objects

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import static java.lang.System.out;

public class ShowRooms {

    public static void main(String args[]) throws
           FileNotFoundException {
        Room rooms[][] = new Room[5][10];
        Scanner myScanner = new Scanner(new
           File("RoomList"));

        for (int floor = 0; floor < 5; floor++) {
            for (int roomNum = 0; roomNum < 10; roomNum++) {
                rooms[floor][roomNum] = new Room();
                rooms[floor][roomNum].readRoom(myScanner);
            }
        }

        for (int floor = 4; floor >= 0; floor--) {
```

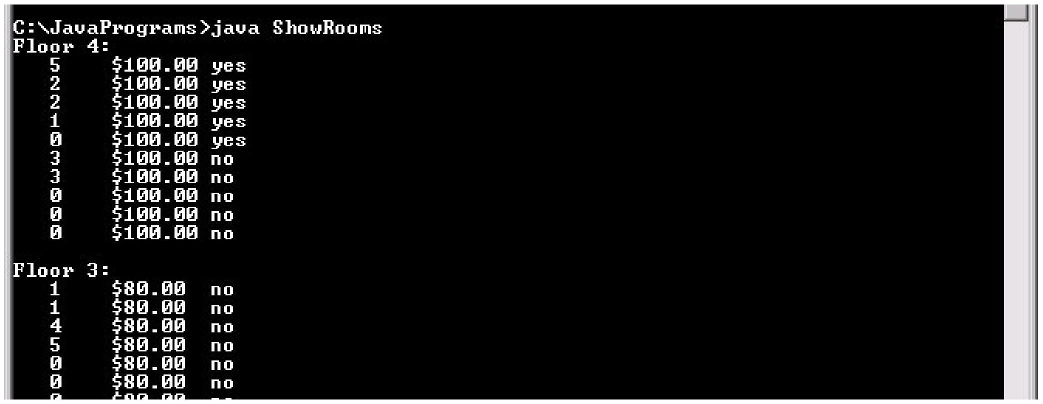
```
                    out.println("Floor " + floor + ":");
                    for (int roomNum = 0; roomNum < 10; roomNum++) {
                        out.print("   ");
                        rooms[floor][roomNum].writeRoom();
                    }
                    out.println();
                }
            }
        }
```

By the time you're done, the program that uses objects is actually simpler than the code that doesn't use objects. That's because, in writing the code with an array of objects, you're taking advantage of methods that are already written as part of the Room class, such as readRoom and writeRoom.

A run of the code in Listing B-2 displays information about all 50 of the hotel's rooms. Instead of showing you all that stuff, Figure B-3 shows you the first several lines in the run. (You don't need to know about every room in the Java Hotel anyway.) The input to the code in Listing B-2, the RoomList file, looks just like the stuff in Listing 11-7. The only difference is that the RoomList file for this section's code has 150 lines in it.

OnTheWeb

You can snare a 150-line RoomList file along with this document's code listings from the book's Web site..



```
C:\JavaPrograms>java ShowRooms
Floor 4:
    5    $100.00 yes
    2    $100.00 yes
    2    $100.00 yes
    1    $100.00 yes
    0    $100.00 yes
    3    $100.00 no
    3    $100.00 no
    0    $100.00 no
    0    $100.00 no
    0    $100.00 no

Floor 3:
    1    $80.00  no
    1    $80.00  no
    4    $80.00  no
    5    $80.00  no
    0    $80.00  no
    0    $80.00  no
```

Figure B-3: Starting a run of the code from Listing B-2.

With all the examples building up to Listing B-2, the code in the listing may be fairly uneventful. The only thing you need to notice is that the line

```
 rooms[floor][roomNum] = new Room();
```

is absolutely, indubitably, 100-percent required. When you accidentally leave off this line (not "if you leave off this line," but "when you leave off this line”), you get a runtime error message saying `java.lang.NullPointerException`.