## Special Topic 7.3

### Two-Dimensional Arrays with Variable Row Lengths

When you declare a two-dimensional array with the command

```
int[][] a = new int[3][3];
```

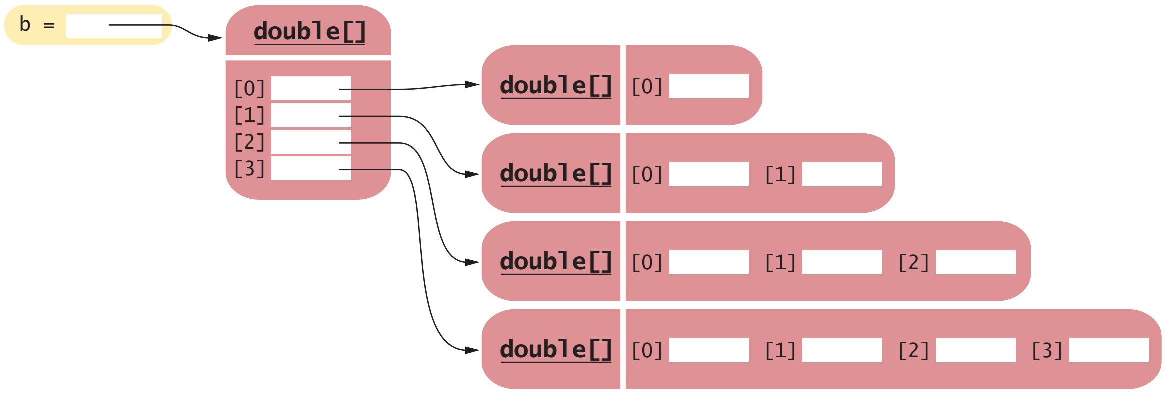
then you get a 3-by-3 matrix that can store 9 elements:

```
a[0][0] a[0][1] a[0][2]
a[1][0] a[1][1] a[1][2]
a[2][0] a[2][1] a[2][2]
```

In this matrix, all rows have the same length.

In Java it is possible to declare arrays in which the row length varies. For example, you can store an array that has a triangular shape, such as:

```
b[0][0]
b[1][0] b[1][1]
b[2][0] b[2][1] b[2][2]
```



A Triangular Array

To allocate such an array, you must work harder. First, you allocate space to hold three rows. Indicate that you will manually set each row by leaving the second array index empty:

```
int[][] b = new int[3][];
```

Then allocate each row separately.

```
for (int i = 0; i < b.length; i++)
   b[i] = new int[i + 1];
```

See the figure.

You can access each array element as `b[i][j]`. The expression `b[i]` selects the `i`th row, and the `[j]` operator selects the `j`th element in that row.

Note that the number of rows is `b.length`, and the length of the `i`th row is `b[i].length`. For example, the following pair of loops prints a ragged array:

```
for (int i = 0; i < b.length; i++)
{
   for (int j = 0; j < b[i].length; j++)
      System.out.print(b[i][j]);
   System.out.println();
}
```

Alternatively, you can use two “for each” loops:

```
for (double[] row : b)
{
   for (double element : row)
      System.out.print(element);
   System.out.println();
}
```

Naturally, such “ragged” arrays are not very common.

Java implements plain two-dimensional arrays in exactly the same way as ragged arrays: as arrays of one-dimensional arrays. The expression `new int[3][3]` automatically allocates an array of three rows, and three arrays for the rows’ contents.