# 2-D Arrays

## - Example

# Two-Dimensional Arrays

- 2-D arrays are used to represent tables, matrices, game boards, images, etc.
- An element of a 2-D array is addressed using a pair of indices, “row” and “column.” For example:

```
board [ r ] [ c ] = 'x';
```

# 2-D Arrays: Declaration

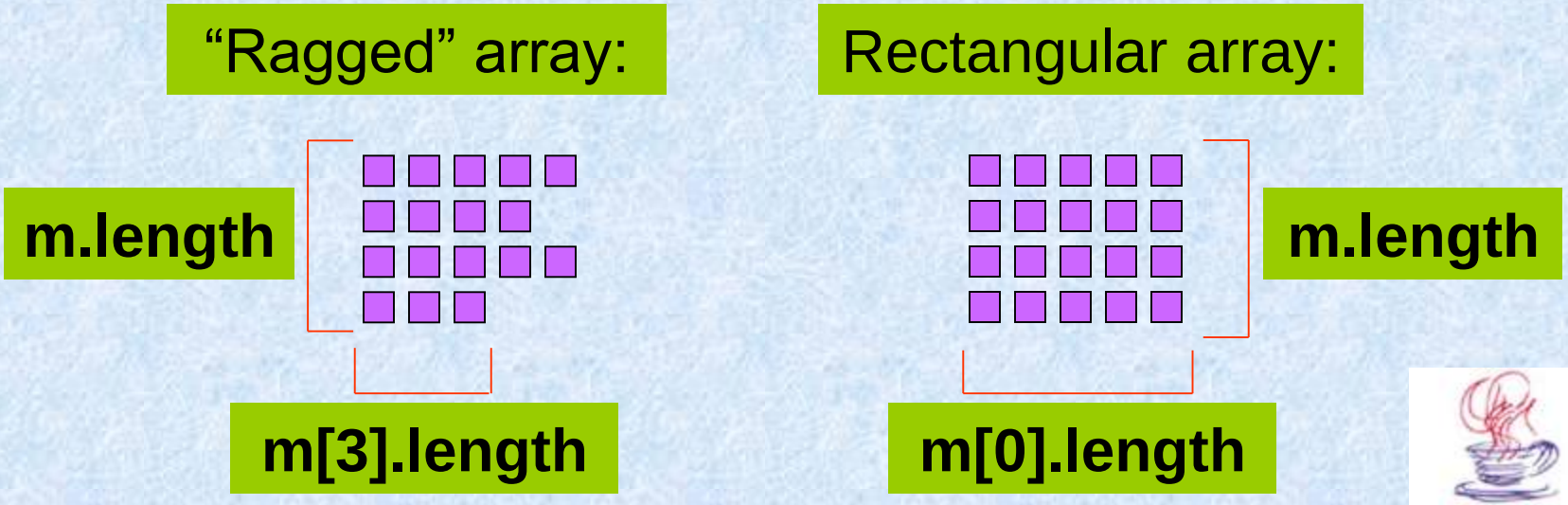
```
// 2-D array of char with 5 rows, 7 cols:
char[ ] [ ] letterGrid = new char [5][7];

// 2-D array of Color with 1024 rows, 768 cols:
Color[ ] [ ] image = new Color [1024][768];

// 2-D array of double with 2 rows and 3 cols:
double [ ] [ ] sample =
    { { 0.0, 0.1, 0.2 },
      { 1.0, 1.1, 1.2 } };
```

# 2-D Arrays: Dimensions

- In Java, a 2-D array is basically a 1-D array of 1-D arrays, its rows. Each row is stored in a separate block of consecutive memory locations.
- If m is a 2-D array, then m[k] is a 1-D array, the *k*-th row.
- m.length is the number of rows.
- m[k].length is the length of the k-th row.

# Dimensions (cont'd)

- Java allows "ragged" arrays, in which different rows have different lengths.
- In a rectangular array, m[0].length can be used to represent the number of columns.



# 2-D Arrays and Nested Loops

- A 2-D array can be traversed using nested loops:

```
for (int  r = 0;  r < m.length;  r++)
{
    for (int  c = 0;  c < m[r].length;  c++)
    {
        ...   //  process m[ r ][ c ]
    }
}
```

# Example

**Matrix Class**

Functionality:

1) Add (another Matrix)
2) Subtract (another Matrix)
3) Multiply (another Matrix)
4) Check for equality (to another Matrix)
5) toString()
6) Necessary Accessors