CSE 142 Summer 2001

2-D Arrays

Introduction

- Review:
  - Simple Arrays
- Today:
  - 2-D arrays
  - Array traversals and processing

Review – Arrays in Java

- Simple, ordered collections
- Elements of a particular array all have the same type
- Size fixed when array created

```
Rectangle[ ] rects = new Rectangle[42+17];
```

- Indexed (direct) access to elements

```
rects[3] = new Rectangle( );
rects[3].move(10, 20);
```


2-D Arrays

- Suppose we want to represent a picture
  (Disclaimer: simple-minded representation for lecture purposes.)
- Want a rectangular array of Colors
- Declaration & creation

```
static final int NROWS = 8;
static final int NCOLS = 8;
Color[ ][ ] picture = new Color[NROWS][NCOLS];
```

- Similar to 1-D array, but we specify two dimensions

2-D Arrays

- Draw the picture

```
static final int NROWS = 8;
static final int NCOLS = 8;
Color[ ][ ] picture = new Color[NROWS][NCOLS];
```


Array Element Access

- Terms: *rows* and *columns*
- Traditionally, the first subscript is the row #, the second is the column #

```
picture[3][2] = Color.red;
picture[1][0] = Color.blue;
picture[4][4] = Color.green;
```

  - Which array elements do these refer to?
  - (Aside: There are applications where it makes sense to think of first subscript as horizontal position and second subscript as vertical, but we'll stick with rows/columns for now.)

Real Representation of 2-D Arrays

- Actual representation in Java* is an array of 1-D arrays
- When we write

```
double[ ][ ] grid = new double[3][5];
```

Java actually does this

```
double[ ][ ] grid = new double[3][ ];
for (int k = 0; k < 3; k++) {
    grid[k] = new double[5];
}
```

- Draw the picture

*In languages like FORTRAN and C/C++, 2-D arrays are really just a grid

Does the Actual Representation Matter?

- Normally no – Most of the time we think of a 2-D array as a rectangular grid
- Main reason to be aware of this is row and column lengths:

```
double[ ][ ] grid = new double[3][5];

grid.length => 3        // # of rows (# elements in 1st dimension)
grid[0].length => 5     // length of row 1
```

- Obvious question: Can rows have different lengths?
- (Maybe not obvious) answer:

2-D Array Traversal

- Typical traversal is to go through the rows and, for each row, go through the columns (known as row-major order; column-major order is another possibility)
- Example of row-major order:

```
// Set all Colors in picture to Color.Red
public void makeRed(Color[ ][ ] picture) {
    for (int row = 0; row < picture.length; row++) {
        for (int col = 0; col < picture[row].length; col++) {
            picture[row][col] = Color.red;
        }
    }
}
```


2-D Array Traversal - Trace

- Trace it!

```
// Set all Colors in picture to Color.Red
public void makeRed(Color[ ][ ] picture)
{
    for (int row = 0;
        row < picture.length; row++) {
      for (int col = 0;
          col < picture[row].length;
                          col++) {
            picture[row][col] =
                          Color.red;
    }
  }
}
```


Exercise: Shift Picture to Left

```
// Slide picture to the left 1 column; fill last column with Color.white
public void makeRed(Color[ ][ ] picture) {
    for (int row = 0; row < _____ ; row++) {
        for (int col = 0; col < _____ ; col++) {
            picture[row][col] = _____;

        }

    }

}
```


Exercise: Shift Picture Down

```
// Slide picture down 1 column; fill first row with Color.black
public void makeRed(Color[ ][ ] picture) {









}
```

- Hint: row-major order might not be the right approach