Arrays of Arrays

An array can represent a collection of any type of object - including other arrays!

The world is filled with examples
- Monthly magazine: we number
  - the monthly editions
  - pages with in each
- Calendars: we number
  - the months
  - days in each month

General Two-Dimensional Arrays

Say we want to develop an annual calendar manager

Representing the Data
- A month is an array of strings that represent daily events
- A year is a 12- element array of months.

Declaring an Array of Arrays

- A month is an array of daily event Strings
- A year is an array of months
So a year is an array of String arrays

private String[ ][ ] dailyEvent;

Creating an Array of Arrays

Array declaration introduces a name, but does not create an array

Proceed in two steps
1. Construct 12- element year
2. Construct each individual month array
1. Construct months

```java
for (int month = 0; month < 12; month++) {
    int numDays = getDays( month+1 );
    dailyEvent[month] = new String[numDays];
}
```

Assume `getDays` is a private method that returns the number of days in a month.

---

### Indexing an Array of Arrays

Say a user enters the information:
- 1/28 - Spring semester starts

The month is 1
- The day is 28
- The event is “Spring semester starts”

Since array indexing begins at 0,
- `dailyEvent[0][27] = “Spring semester starts”;

---

### Setting and Getting Array Values

```java
// Set the event description for a given month and day
public void setEvent( int month, int day, String description ) {
    dailyEvent[month-1][day-1] = description;
}

// Returns the event associated with a given date
public String getEvent( int month, int day ) {
    return dailyEvent[month-1][day-1];
}
```

---

### Arrays of Arrays are two dimensional

When you think of an array of arrays in this way, it is natural to think of indices as specifying row and column:
- `someArray[rowNum][colNum]`
Traversing a 2-D Array

Often want to do something with every element in an array—Use for loops!

- Ex. Initialize all calendar entries to “No event today”
  - to initialize all calendar entries for a single month:
    ```java
    for (int day = 0; day < dailyEvent[month].length; day++) {
        dailyEvent[month][day] = “No event today”;
    }
    ```
  - to initialize all 12 months
    ```java
    // Fill all entries in each month with “No event today”
    for (int month = 0; month < 12; month++) {
        // Fill all entries for one month with “No event today”
        ...
    }
    ```

General Structure of Nested for Loops for 2-D Arrays

```java
for (row = 0; row < myArray.length; row++) {
    for (col = 0; col < myArray[row].length; col++) {
        // Do something with array element myArray[row][col]
        ...
    }
}
```

Beyond Two Dimensions

- so far our yearly calendar is quite limited
- might want a day’s entries to include
  - daily events
  - hourly events

A CalendarDay Class

Two arrays
- full- day events
- hourly events
  ```java
  private String[] fullDayEvent;
  private String[] hourlyEvent;
  ```

A year is then just an array of months, each of which is an array of CalendarDay, each of which is two arrays

Putting it all Together

```java
// Fill all entries in each month with “No event today”
for (int month = 0; month < 12; month++) {
    // Fill all entries in each month with “No event today”
    for (int day = 0; day < dailyEvent[month].length; day++) {
        dailyEvent[month][day] = “No event today”;
    }
}
```

Exercises: Add the Following to CalendarDay

```java
// Clear all event entries for this day
public void clearDay();
```

```java
// Return the event for a specific hour on this day
public String getHourlyEvent(int time);
```

```java
// Print all hourly events:
public void printHourlyEvents();
```

```java
// Each hourly event on a separate line in the form
// hour: 00 event
public void printHourlyEvents();
```

```java
// Print all full- day events
public void printFullDayEvents();
```

```java
// Print all events for this day
public void printDay();
```
Matrices

- two dimensional arrays with rows of same length
  Ex. magnified region of pixels from an image

Each pixel can be described by row and column position, as well as color value

More examples

- chessboards
- sliding block puzzles

Magic Square

- a matrix in which the sums of rows, columns, and diagonals are all equal

```
4 9 2
3 5 7
8 1 6
```

```
11 18 25 2 9
10 12 19 21 3
4 5 13 20 22
23 5 7 14 16
17 24 1 8 15
```

Declaring and Constructing a Matrix

- Matrices are simply 2-D arrays, so a matrix is declared in the same way

```
private int[][] magicSquare;
```

- Matrix must be constructed before it is filled

```
magicSquare = new int[SIZE][SIZE];
```

- `n`-row, `m`-column matrix constructed as follows

```
rectangularArray = new type[n][m];
```

Traversing a Matrix

Ex. Determine whether a square matrix is a magic square

- Row, column, and diagonal sums must be equal.
- Start by finding target sum

```
// Compute sum of elements in row 0
int targetSum = 0;
for (int col = 0; col < SIZE; col++) {
    targetSum = targetSum + magicSquare[0][col];
}
```

Row by Row Traversal

- check sum of each row
- use nested for loops!

```
// Assume we have a magic square unless a sum is incorrect
boolean isMagicSquare = true;
for (int row = 1; row < SIZE; row++) {
    if (Check sum of each row
        int sum = 0;
        for (col = 0; col < SIZE; col++) {
            sum = sum + magicSquare[row][col];
        }
        if (sum != targetSum) {
            isMagicSquare = false;
            return;
        }
    } else {
        isMagicSquare = false;
        return;
    }
```
A More Efficient Version

- If any row’s sum does not match target, can stop right away
  
  ```java
  boolean isMagicSquare = true;
  for (int row = 1; row < SIZE && isMagicSquare; row++) {
      int sum = 0;
      for (col = 0; col < SIZE; col++) {
          sum = sum + magicSquare[row][col];
      }
      if (sum != targetSum) {
          isMagicSquare = false;
      }
  }
  ```

Column by Column

- nested loops again
- reverse order of nesting
  - outer loop through columns
  - inner loop through rows

```java
boolean isMagicSquare = true;
for (int col = 0; col < SIZE && isMagicSquare; col++) {
    int sum = 0;
    for (row = 0; row < SIZE; row++) {
        sum = sum + magicSquare[row][col];
    }
    isMagicSquare = (sum == targetSum);
}
```

Diagonal Traversal

- two diagonals- two loops
- no nested loops this time
  
  ```java
  int sum = 0;
  for (int element = 0; element < SIZE; element++) {
      sum = sum + magicSquare[element][element];
  }
  isMagicSquare = (sum == targetSum);
  ```

Minor Diagonal

- a bit more tricky to get indices right
- for a 4x4 matrix:
  - [0][3], [1][2], [2][1], [3][0]
  - if loop var is row (over 0,1,2,3), associated column is (SIZE-1)-row
  
  ```java
  int sum = 0;
  for (row = 0; row < SIZE; row++) {
      sum = sum + magicSquare[row][SIZE-1-row];
  }
  isMagicSquare = (sum == targetSum);
  ```

Array Indexing

- Now let’s fill a magic square
- Through this we’ll explore more complex ways to “walk through” matrices
  
  ```java
  // Check sum of major diagonal
  int sum = 0;
  for (int element = 0; element < SIZE; element++) {
      sum = sum + magicSquare[element][element];
  }
  isMagicSquare = (sum == targetSum);
  ```

Filling a Magic Square

For an n-by-n matrix, if n is odd, a simple algorithm will fill it appropriately

- place a 1 in the center of the bottom row. Then fill in the remainder of the square by following these rules:
  - Try to place the next integer (one greater than the last one you placed) in the cell one slot below and to the right of the last one placed. If you fall off the bottom of the array, go to the top row of the same column. If you fall off the right edge of the array, go to the leftmost column. If that cell is empty, write the next integer there and continue. If the cell is full, go back to where you wrote the last integer and write the next one in the cell directly above it.
Initializing Matrix Entries to a Value

Begin by initializing each cell in a matrix to 0

```java
// Initialize all entries in the square to 0
for (int row = 0; row < SIZE; row++) {
    for (int col = 0; col < SIZE; col++) {
        magicSquare[row][col] = 0;
    }
}
```

Filling the Magic Square

- Declare and initialize variables to keep track of current row and column
  ```java
  int currRow = SIZE - 1;
  int currCol = SIZE / 2;
  ```

- Assuming SIZE is odd!

Follow the Rules

- will follow the rules for filling a magic square
- apply to the rules until the square is filled (SIZE x SIZE times)

```java
for (int nextInt = 1; nextInt <= SIZE * SIZE; nextInt++) {
    // fill a cell
}
```
In the Loop

- Fill the current cell

\[
magicSquare[currRow][currCol] = nextInt;
\]

- Find the next cell to fill

Finding the Next Cell

Move down and to the right

\[
\begin{align*}
nextCol &= currCol + 1; \\
nextRow &= currRow + 1;
\end{align*}
\]

Might "fall off" the matrix!

Need to wrap around

\[
\begin{align*}
nextCol &= (currCol + 1) \mod SIZE; \\
nextRow &= (currRow + 1) \mod SIZE;
\end{align*}
\]

Adjusting if a Cell is Full

If a cell is already filled, go back to previous cell and move up

\[
\text{if (magicSquare[nextRow][nextCol] == 0) } \\
\hspace{1cm} \text{// Use the cell if it is available} \\
\hspace{1cm} \text{currRow = nextRow; } \\
\hspace{1cm} \text{currCol = nextCol; } \\
\text{else } \text{// Move up one cell otherwise} \\
\hspace{1cm} \text{currRow = (currRow - 1 + SIZE) \mod SIZE;}
\]

More Practice with Matrices: Digital Image Smoothing

- A digital photo is a matrix of pixel values
- Noise and other artifacts sometimes appear
- can smooth an image to eliminate (or decrease) some of these artifacts

A Simple Smoothing Algorithm

- Assume image is black and white
  - pixel values are ints giving brightness levels

\[
\text{private int}[ ][ ] \text{ brightness;}
\]

- Implement smoothing by averaging
  - replace each pixel value with the average of its neighbors

Design of Smoothing Algorithm

- Assume constants WIDTH and HEIGHT give image width and height in pixels
- Declare a matrix for the averaged (smoothed) image

\[
\text{private int}[ ][ ] \text{ avgBrightness;}
\]

- Traverse row by row
  - replace brightness value by average value
Calculating Average Brightness

Consider a 3-by-3 square centered at the pixel to be replaced by an average.

```java
// Calculate average brightness at position specified by row and col;
// position is not on an edge of the image.
int totalBrightness = 0;
for (int r = -1; r <= 1; r++) {
    for (int c = -1; c <= 1; c++) {
        totalBrightness += brightness[row+r][col+c];
    }
    avgBrightness[row][col] = totalBrightness/9;
}
```

Pixels on the Edge

What about pixels on the edges of an image?
- pixels on top edge have no neighbors above them; only to the sides and below.
- pixels on bottom have no neighbors below them; only to the sides and above.
- pixels in the corners have even fewer neighbors.

Handling the Top Edge

```java
// Calculate average brightness at position specified by row and col;
// position is top edge of the image, not in a corner
int totalBrightness = 0;
for (int r = 0; r <= 1; r++) {
    for (int c = -1; c <= 1; c++) {
        totalBrightness += brightness[row+r][col+c];
    }
    avgBrightness[row][col] = totalBrightness/6;
}
```

Handling the Bottom Edge

```java
// Calculate average brightness at position specified by row and col;
// position is bottom edge of the image, not in a corner
int totalBrightness = 0;
for (int r = -1; r <= 0; r++) {
    for (int c = -1; c <= 1; c++) {
        totalBrightness += brightness[row+r][col+c];
    }
    avgBrightness[row][col] = totalBrightness/6;
}
```

Generalizing

- Similar loops to handle:
  - left border
  - right border
  - corners
- Difference is how many neighbors are found above, below, to the right, and to the left:

  ```java
  // calculate average brightness at position specified by row and col;
  // position is top edge of the image, not in a corner
  int totalBrightness = 0;
  int distUp = 1;
  for (int r = 0; r <= distUp; r++) {
      for (int c = -1; c <= 1; c++) {
          totalBrightness += brightness[row+r][col+c];
      }
      neighbors++; // incremental adjustment
  }
  avgBrightness[row][col] = totalBrightness/neighbors;
  ```

```java
// calculate average brightness at position specified by row and col;
// position is bottom edge of the image, not in a corner
int totalBrightness = 0;
int distDown = 1;
for (int r = -1; r <= 0; r++) {
    for (int c = -1; c <= 1; c++) {
        totalBrightness += brightness[row+r][col+c];
    }
    neighbors++; // incremental adjustment
  }
  avgBrightness[row][col] = totalBrightness/neighbors;
```