CMSC 150
INTRODUCTION TO COMPUTING

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING IN JAVA: AN INTERDISCIPLINARY APPROACH, SEDGEWICK AND WAYNE (PEARSON ADDISON-WESLEY 2007)

LECTURE 4

• ARRAYS

• MULTIDIMENSIONAL ARRAYS
MANY VARIABLES OF THE SAME TYPE

• Goal. 10 variables of the same type.
1. // tedious and error-prone
2. double a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
3. a0 = 0.0;
4. a1 = 0.0;
5. a2 = 0.0;
6. a3 = 0.0;
7. a4 = 0.0;
8. a5 = 0.0;
9. a6 = 0.0;
10. a7 = 0.0;
11. a8 = 0.0;
12. a9 = 0.0;
13. ...
14. a4 = 3.0;
15. ...
16. a8 = 8.0;
17. ...
18. double x = a4 + a8;
MANY VARIABLES OF THE SAME TYPE

• Goal. 10 variables of the same type.
1. // easy alternative
2. `double[] a = new double[10];`
3. ...
4. `a[4] = 3.0;`
5. ...
6. `a[8] = 8.0;`
7. ...
8. `double x = a[4] + a[8];`

declares, creates, and initializes [stay tuned for details]
MANY VARIABLES OF THE SAME TYPE

- Goal. 1 million variables of the same type.
1. // scales to handle large arrays
2. double[] a = new double[1000000];
3. ...
4. a[123456] = 3.0;
5. ...
6. a[987654] = 8.0;
7. ...
8. double x = a[123456] + a[987654];
ARRAYS

• **Array.** Indexed sequence of values of the same type
• Store and manipulate huge quantities of data.
• Examples.
  • 52 playing cards in a deck
  • 3 thousand undergrads at UR
  • 140 characters per Tweet
  • 4 billion nucleotides in a DNA strand
  • 50 trillion cells in the human body
  • $6.022 \times 10^{23}$ particles in a mole

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Captain America</td>
</tr>
<tr>
<td>1</td>
<td>Ironman</td>
</tr>
<tr>
<td>2</td>
<td>Thor</td>
</tr>
<tr>
<td>3</td>
<td>Hulk</td>
</tr>
<tr>
<td>4</td>
<td>Black Widow</td>
</tr>
<tr>
<td>5</td>
<td>Hawkeye</td>
</tr>
<tr>
<td>6</td>
<td>Nick Fury</td>
</tr>
<tr>
<td>7</td>
<td>Wannabe Avengerman</td>
</tr>
</tbody>
</table>
ARRAYS IN JAVA

- Java has special language support for arrays.
  - To make an array: declare, create, and initialize it
  - To access entry $i$ of array named $a$, use $a[i]$
  - Array indices start at 0

1. `int N = 10;` // size of array
2. `double[] a;` // declare the array
3. `a = new double[N];` // create the array
4. `for (int i = 0; i < N; i++)` // initialize the
5. `a[i] = 0.0;` // array all to 0.0

OR

1. `double[] a = new double[10];` // declare, create,
2. // and initialize. Default will initialize to 0 for double.
 VECTOR DOT PRODUCT

• Dot product. Given two vectors $x[]$ and $y[]$ of length $N$, their dot product is the sum of the products of their corresponding components.

1. double[] $x = \{0.3, 0.6, 0.1\}$; //Another way to initialize
2. double[] $y = \{0.5, 0.1, 0.4\}$;
3. int $N = x$.length;
4. double $sum = 0.0$;
5. for (int $i = 0; i < N; i++)$
6.   $sum = sum + x[i]*y[i]$;
7. }

<table>
<thead>
<tr>
<th>$i$</th>
<th>$x[i]$</th>
<th>$y[i]$</th>
<th>$x[i]*y[i]$</th>
<th>$sum$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.30</td>
<td>.50</td>
<td>.15</td>
<td>.15</td>
</tr>
<tr>
<td>1</td>
<td>.60</td>
<td>.10</td>
<td>.06</td>
<td>.21</td>
</tr>
<tr>
<td>2</td>
<td>.10</td>
<td>.40</td>
<td>.04</td>
<td>.25</td>
</tr>
</tbody>
</table>
### ARRAY-PROCESSING EXAMPLES

<table>
<thead>
<tr>
<th>Task</th>
<th>Java Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>create an array with random values</td>
<td><code>double[] a = new double[N]; for (int i = 0; i &lt; N; i++) a[i] = Math.random();</code></td>
</tr>
<tr>
<td>print the array values, one per line</td>
<td><code>for (int i = 0; i &lt; N; i++) System.out.println(a[i]);</code></td>
</tr>
<tr>
<td>find the maximum of the array values</td>
<td><code>double max = Double.NEGATIVE_INFINITY; for (int i = 0; i &lt; N; i++) if (a[i] &gt; max) max = a[i];</code></td>
</tr>
<tr>
<td>compute the average of the array values</td>
<td><code>double sum = 0.0; for (int i = 0; i &lt; N; i++) sum += a[i]; double average = sum / N;</code></td>
</tr>
<tr>
<td>copy to another array</td>
<td><code>double[] b = new double[N]; for (int i = 0; i &lt; N; i++) b[i] = a[i];</code></td>
</tr>
<tr>
<td>reverse the elements within an array</td>
<td><code>for (int i = 0; i &lt; N/2; i++) { double temp = b[i]; b[i] = b[N-1-i]; b[N-i-1] = temp; }</code></td>
</tr>
</tbody>
</table>
EXAMPLES: DECK OF CARDS
SETTING ARRAY VALUES AT COMPILE TIME

• Ex. Print a random card.
  1. String[] rank = {
  2.     "2", "3", "4", "5", "6", "7", "8", "9",
  3.     "10", "Jack", "Queen", "King", "Ace"
  4.   };
  5.  
  6. String[] suit = {
  7.     "Clubs", "Diamonds", "Hearts", "Spades"
  8.   };
  9.  
  10. int i = (int) (Math.random() * 13);    // between 0 and 12
  11. int j = (int) (Math.random() * 4);     // between 0 and 3
  12. 
  13. System.out.println(rank[i] + " of " + suit[j]);
SETTING ARRAY VALUES AT RUN TIME

- Ex. Create a deck of playing cards and print them out.
  1. String[] deck = new String[52]; //Default initialized to ""
  2. for (int i = 0; i < 13; i++) //Reassign the values to something meaningful
  3. for (int j = 0; j < 4; j++)
  4. deck[4*i + j] = rank[i] + " of " + suit[j];
  5. for (int i = 0; i < 52; i++)
  6. System.out.println(deck[i]);

- Q. In what order does it output them?
- A. two of clubs  B. two of clubs
  two of diamonds  three of clubs
  two of hearts  four of clubs
  two of spades  five of clubs
  three of clubs  six of clubs
  ...  ...
SHUFFLING

• Goal. Given an array, rearrange its elements in random order.

• Shuffling algorithm.
  • In iteration $i$, pick random card from deck[$i$] through deck[$N-1$], with each card equally likely.
  • Exchange it with deck[$i$].

1. `int N = deck.length;` //Use .length to know how many elements
2. `for (int i = 0; i < N; i++) {` //there are.
3. `int r = i + (int) (Math.random() * (N-i));` //Random between $i$ and $N - 1$
4. `String t = deck[r];` //Swap
5. `deck[r] = deck[i];`
6. `deck[i] = t;`
7. `}`
SHUFFLING A DECK OF CARDS: PUTTING EVERYTHING TOGETHER

1. public class Deck {
2.      public static void main(String[] args) {
3.         String[] suit = {"Clubs", "Diamonds", "Hearts", "Spades"}; //Define suits, ranks, and sizes
4.         String[] rank = {
6.         }
7.         int SUITS = suit.length, RANKS = rank.length, N = SUITS * RANKS;
8.         String[] deck = new String[N]; //Build deck
9.         for (int i = 0; i < RANKS; i++) {
10.            for (int j = 0; j < SUITS; j++)
11.                deck[SUITS*i + j] = rank[i] + " of " + suit[j];
12.         }
13.         for (int i = 0; i < N; i++) { //Shuffle
14.             int r = i + (int) (Math.random() * (N-i));
15.             String t = deck[r];
16.             deck[r] = deck[i];
17.             deck[i] = t;
18.         }
19.         for (int i = 0; i < N; i++) //Print the deck
20.             System.out.println(deck[i]);
21.     }
22. }
STRINGS REVISITED

• Strings are arrays of `char`! Well sort of…

• Strings ‘underneath the hood’ are arrays of `char`, but we use them differently
  • [Java API for String]
  • Use `charAt(i)` instead of `[i]`
  • Convert to `char[]` using `toCharArray()` and back to a String easily
    String `s` = “Hello”;
    `char[] c` = `s.toCharArray()``;
    String `s2` = new String(`c``);
  • Allows more, e.g., `substring()` which returns a portion of the String
EXERCISE – PARTNERS

• Write an algorithm/program to find the minimum, maximum, and average of an array of doubles. Use a single loop!
MULTIDIMENSIONAL ARRAYS
TWO-DIMENSIONAL ARRAYS

- Two-dimensional arrays.
  - Table of data for each experiment and outcome.
  - Table of grades for each student and assignments.
  - Pixels in an image


- Java abstraction. 2D array.
TWO-DIMENSIONAL ARRAYS IN JAVA

• Array access. Use $a[i][j]$ to access entry in row $i$ and column $j$.

• Zero-based indexing. Row and column indices start at 0.

1. `int M = 10, N = 3;`
2. `double[][] a = new double[M][N];`
3. `for (int i = 0; i < M; i++)`
4. `for (int j = 0; j < N; j++)`
5. `a[i][j] = 0.0;`
• Initialize 2D array by listing values.
1. `double[][] p = {
2.   {.02, .92, .02, .02, .02 },
3.   {.02, .02, .32, .32, .32 },
4.   {.02, .02, .02, .92, .02 },
5.   {.92, .02, .02, .02, .02 },
6.   {.47, .02, .47, .02, .02 },
7.   };`
MATRIX ADDITION

- Matrix addition. Given two N-by-N matrices a and b, define c to be the N-by-N matrix where \( c[i][j] \) is the sum \( a[i][j] + b[i][j] \)

1. `double[][] c = new double[N][N];`
2. `for (int i = 0; i < N; i++)`
3. `for (int j = 0; j < N; j++)`
4. `c[i][j] = a[i][j] + b[i][j];`
Matrix multiplication. Given two N-by-N matrices \( a \) and \( b \), define \( c \) to be the N-by-N matrix where \( c[i][j] \) is the dot product of the \( i \)th row of \( a[i][] \) and the \( j \)th column of \( b[][j] \).

1. `double[][] c = new double[N][N];`
2. `for (int i = 0; i < N; i++)`
3. `for (int j = 0; j < N; j++)`
4. `for (int k = 0; k < N; k++)`
5. `c[i][j] += a[i][k] * b[k][j];`
ODDITIES OF MULTI-DIMENSIONAL ARRAYS

• A multidimensional array is considered “ragged” if the columns do not have equal lengths.

1. `int N = 10;`
2. `int[][] ragged = new int[N][];`
3. `for(int i = 1; i <= N; ++i)`
4. `ragged[i-1] = new int[i];`
EXERCISE – PARTNERS

• Write an algorithm/program to transpose a Matrix. Transposing means that each row becomes a column in a new matrix.
SUMMARY

• Arrays.
  • Organized way to store huge quantities of data.
  • Almost as easy to use as primitive types.
  • Can directly access an element given its index.

http://imgs.xkcd.com/comics/donald_knuth.png