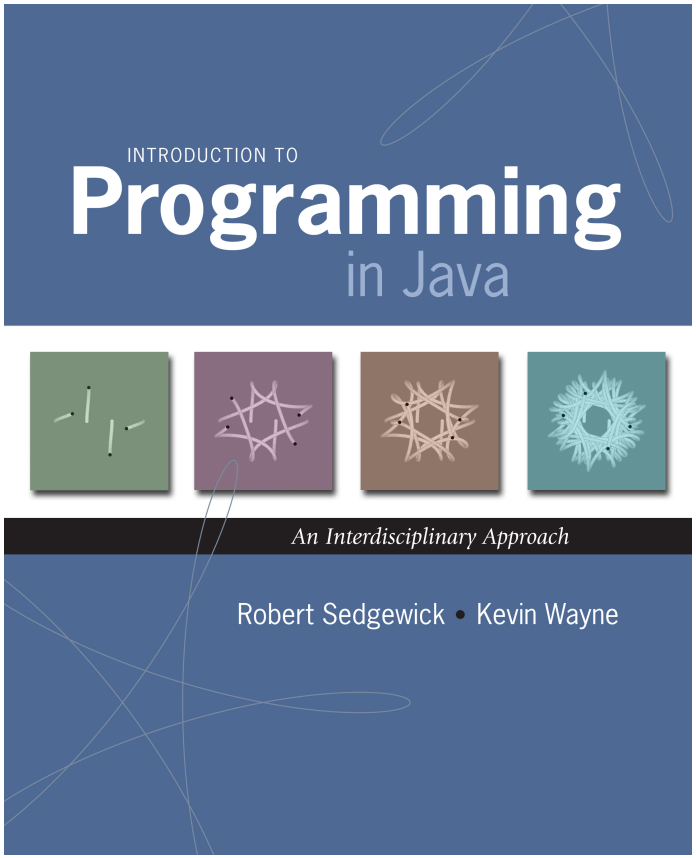
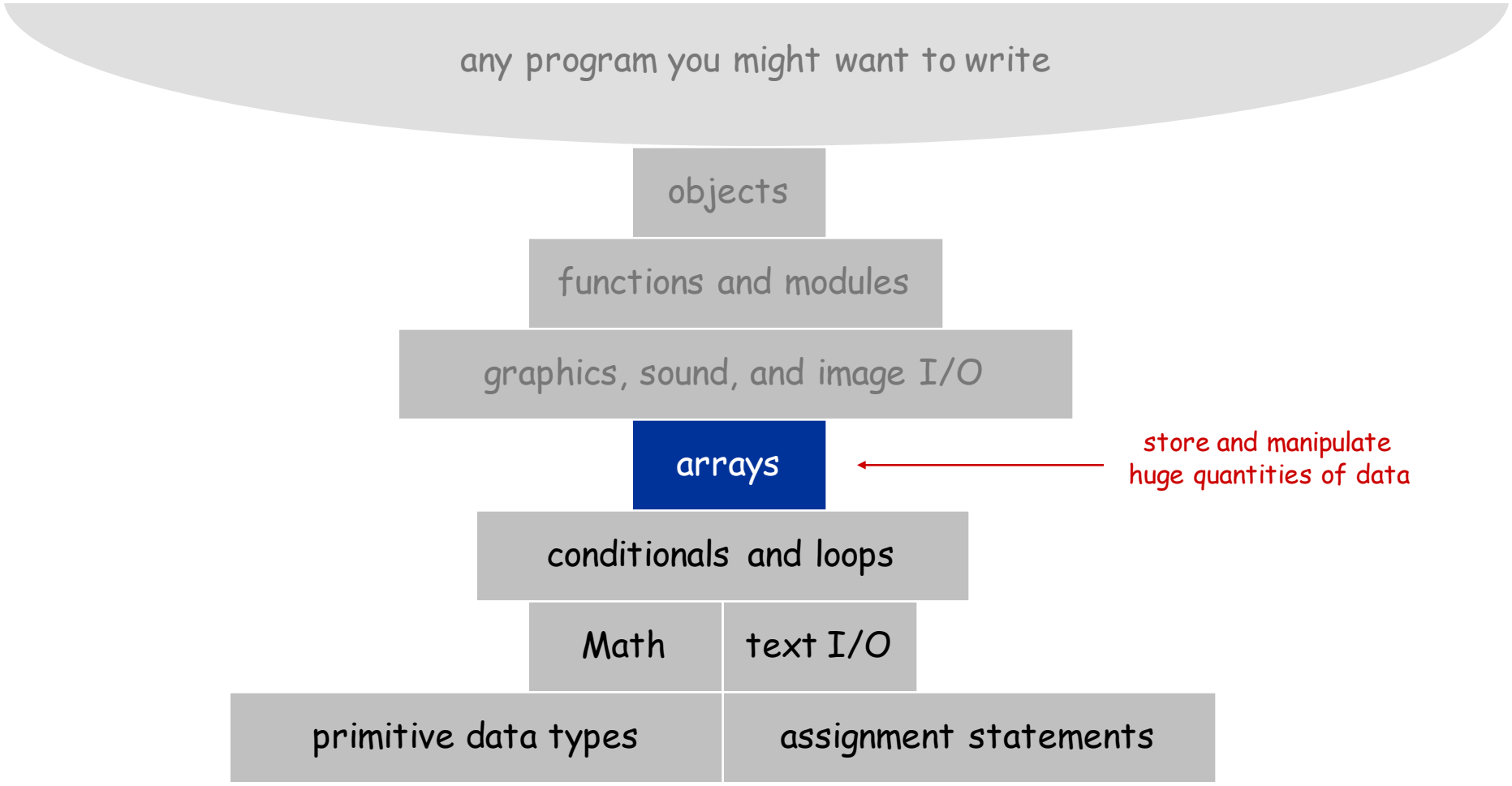
1.4 Arrays


INTRODUCTION TO
Programming
in Java
An Interdisciplinary Approach
Robert Sedgewick • Kevin Wayne

A Foundation for Programming



Arrays

This lecture. Store and manipulate huge quantities of data.

Array. Indexed sequence of values of the same type.

Examples.

- 52 playing cards in a deck.
- 5 thousand undergrads at Princeton.
- 1 million characters in a book.
- 10 million audio samples in an MP3 file.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 50 trillion cells in the human body.
- 6.02×10^{23} particles in a mole.

index	value
0	wayne
1	rs
2	doug
3	dgabai
4	maia
5	llp
6	funk
7	vertanen

Many Variables of the Same Type

Goal. 10 variables of the same type.

```
// tedious and error-prone
double a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
a0 = 0.0;
a1 = 0.0;
a2 = 0.0;
a3 = 0.0;
a4 = 0.0;
a5 = 0.0;
a6 = 0.0;
a7 = 0.0;
a8 = 0.0;
a9 = 0.0;
…
a4 = 3.0;
…
a8 = 8.0;
…
double x = a4 + a8;
```

Many Variables of the Same Type

Goal. 10 variables of the same type.

```
// easy alternative
double[] a = new double[10];
…
a[4] = 3.0;
…
a[8] = 8.0;
…
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.

```
// scales to handle large arrays
double[] a = new double[1000000];
…
a[123456] = 3.0;
…
a[987654] = 8.0;
…
double x = a[123456] + a[987654];
```

declares, creates, and initializes
[stay tuned for details]

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`.

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

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`.

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

Compact alternative.

- Declare, create, and initialize in one statement.
- Default initialization: all numbers automatically set to zero.

```
int N = 10;                       // size of array
double[] a = new double[N];       // declare, create, init
```

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.

```
double[] x = { 0.3, 0.6, 0.1 };
double[] y = { 0.5, 0.1, 0.4 };
int N = x.length;
double sum = 0.0;
for (int i = 0; i < N; i++) {
   sum = sum + x[i]*y[i];
}
```

i	x[i]	y[i]	x[i]*y[i]	sum
				0
0	.30	.50	.15	.15
1	.60	.10	.06	.21
2	.10	.40	.04	.25
				.25

Array-Processing Examples

create an array with random values	`double[] a = new double[N];`
	`for (int i = 0; i < N; i++)`
	`  a[i] = Math.random();`

print the array values, one per line	`for (int i = 0; i < N; i++)` `  System.out.println(a[i]);`

find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < N; i++)`
	`  if (a[i] > max) max = a[i];`

compute the average of the array values	`double sum = 0.0;`
	`for (int i = 0; i < N; i++)`
	`  sum += a[i];`
	`double average = sum / N;`

copy to another array	`double[] b = new double[N];`
	`for (int i = 0; i < N; i++)`
	`  b[i] = a[i];`

reverse the elements within an array	`for (int i = 0; i < N/2; i++)`
	`{`
	`  double temp = b[i];`
	`  b[i] = b[N-1-i];`
	`  b[N-i-1] = temp;`
	`}`

Shuffling a Deck

Setting Array Values at Compile Time

Ex. Print a random card.

```
String[] rank = {
   "2", "3", "4", "5", "6", "7", "8", "9",
   "10", "Jack", "Queen", "King", "Ace"
};

String[] suit = {
   "Clubs", "Diamonds", "Hearts", "Spades"
};

int i = (int) (Math.random() * 13);   // between 0 and 12
int j = (int) (Math.random() * 4);    // between 0 and 3

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.

```
String[] deck = new String[52];
for (int i = 0; i < 13; i++)
   for (int j = 0; j < 4; j++)
      deck[4*i + j] = rank[i] + " of " + suit[j];

for (int i = 0; i < 52; i++)
   System.out.println(deck[i]);
```

typical array-processing code changes values at runtime

Q. In what order does it output them?

A.
```
two of clubs
two of diamonds
two of hearts
two of spades
three of clubs
...
```

B.
```
two of clubs
three of clubs
four of clubs
five 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]`.

```
int N = deck.length;
for (int i = 0; i < N; i++) {
   int r = i + (int) (Math.random() * (N-i));
   String t = deck[r];
   deck[r] = deck[i];
   deck[i] = t;
}
```

swap idiom

between i and N-1

Shuffling a Deck of Cards: Putting Everything Together

```
public class Deck {
   public static void main(String[] args) {
      String[] suit = { "Clubs", "Diamonds", "Hearts", "Spades" };
      String[] rank = { "2", "3", "4", "5", "6", "7", "8", "9",
                        "10", "Jack", "Queen", "King", "Ace"    };
      int SUITS = suit.length;
      int RANKS = rank.length;
      int N = SUITS * RANKS;

      String[] deck = new String[N];                          build the deck
      for (int i = 0; i < RANKS; i++)
         for (int j = 0; j < SUITS; j++)
            deck[SUITS*i + j] = rank[i] + " of " + suit[j];

      for (int i = 0; i < N; i++) {                                shuffle
         int r = i + (int) (Math.random() * (N-i));
         String t = deck[r];
         deck[r] = deck[i];
         deck[i] = t;
      }

      for (int i = 0; i < N; i++)                          print shuffled deck
         System.out.println(deck[i]);
   }
}
```

avoid "hardwired" constants

Shuffling a Deck of Cards

```
% java Deck
5 of Clubs
Jack of Hearts
9 of Spades
10 of Spades
9 of Clubs
7 of Spades
6 of Diamonds
7 of Hearts
7 of Clubs
4 of Spades
Queen of Diamonds
10 of Hearts
5 of Diamonds
Jack of Clubs
Ace of Hearts
...
5 of Spades
```

```
% java Deck
10 of Diamonds
King of Spades
2 of Spades
3 of Clubs
4 of Spades
Queen of Clubs
2 of Hearts
7 of Diamonds
6 of Spades
Queen of Spades
3 of Spades
Jack of Diamonds
6 of Diamonds
8 of Spades
9 of Diamonds
...
10 of Spades
```

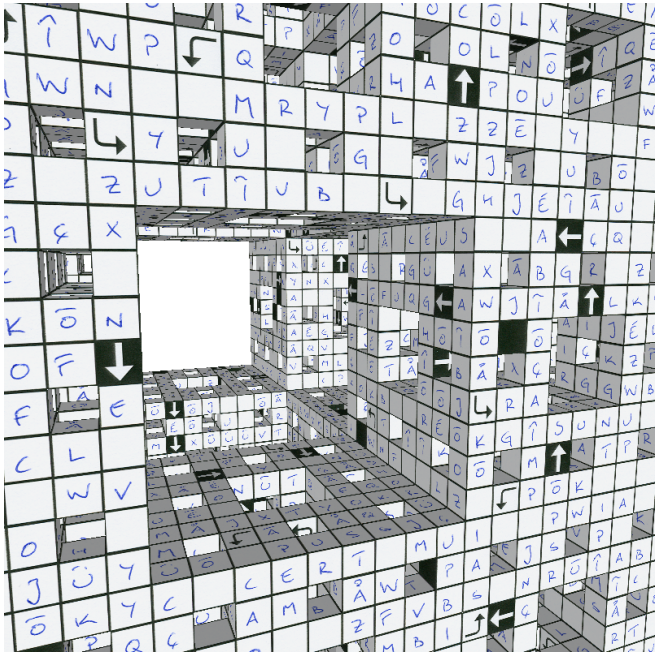
War Story (PlanetPoker.com)

Texas hold 'em poker. Software must shuffle electronic deck of cards.

How we learned to cheat at online poker: a study in software security

`http://itmanagement.earthweb.com/entdev/article.php/616221`

Multidimensional Arrays



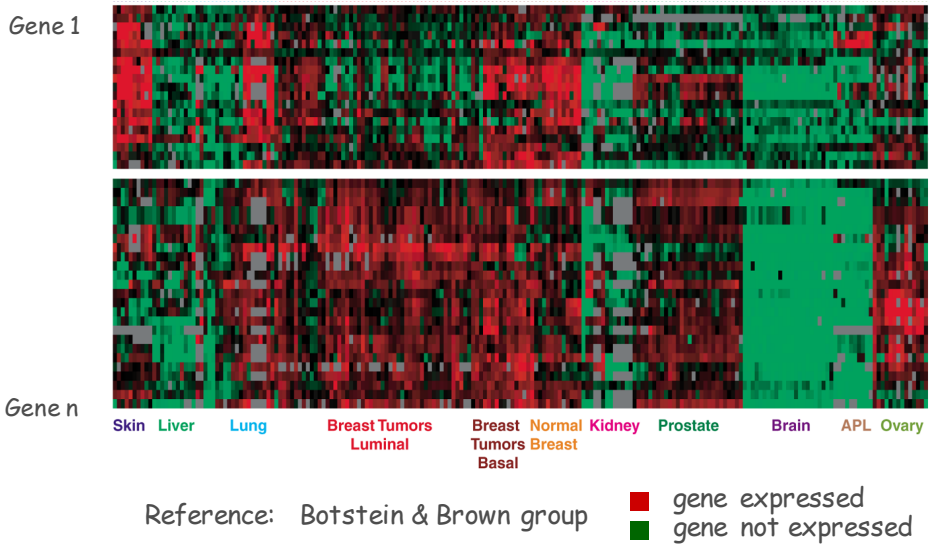
Two-Dimensional Arrays

Two-dimensional arrays.

- Table of data for each experiment and outcome.
- Table of grades for each student and assignments.
- Table of grayscale values for each pixel in a 2D image.

Mathematical abstraction. Matrix.

Java abstraction. 2D array.

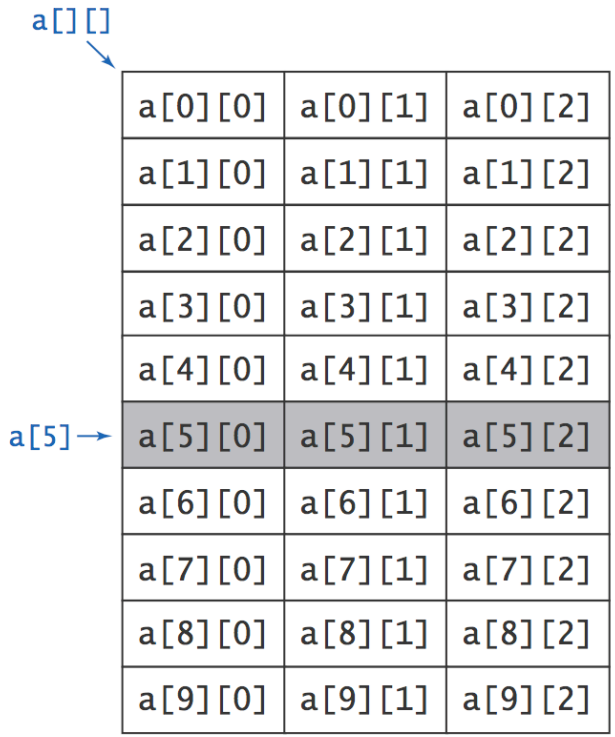


Two-Dimensional Arrays in Java

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`.

```
int M = 10;
int N = 3;
double[][] a = new double[M][N];
for (int i = 0; i < M; i++) {
   for (int j = 0; j < N; j++) {
      a[i][j] = 0.0;
   }
}
```

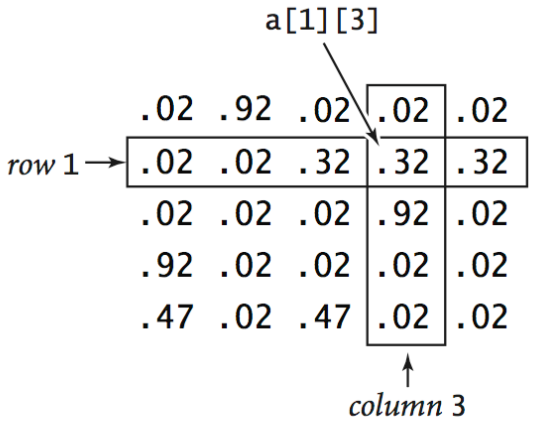


A 10-by-3 array

Setting 2D Array Values at Compile Time

Initialize 2D array by listing values.

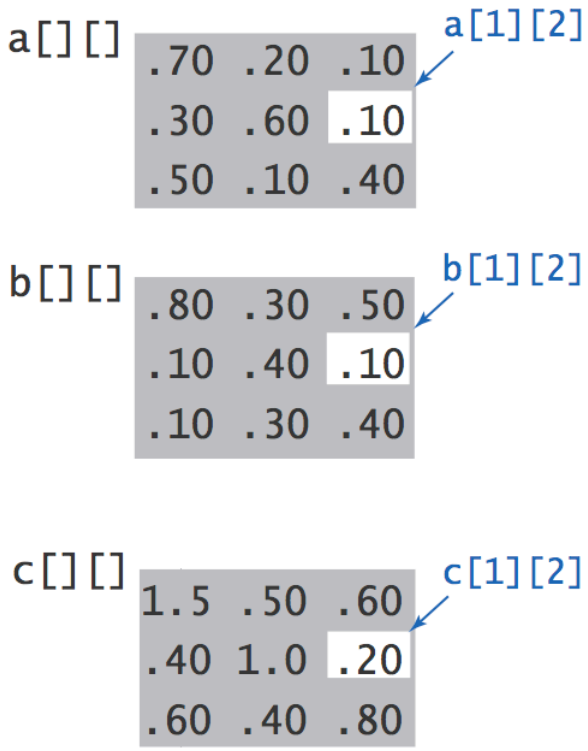
```
double[][] p = {
    { .02, .92, .02, .02, .02 },
    { .02, .02, .32, .32, .32 },
    { .02, .02, .02, .92, .02 },
    { .92, .02, .02, .02, .02 },
    { .47, .02, .47, .02, .02 },
};
```



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]`.

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      c[i][j] = a[i][j] + b[i][j];
```



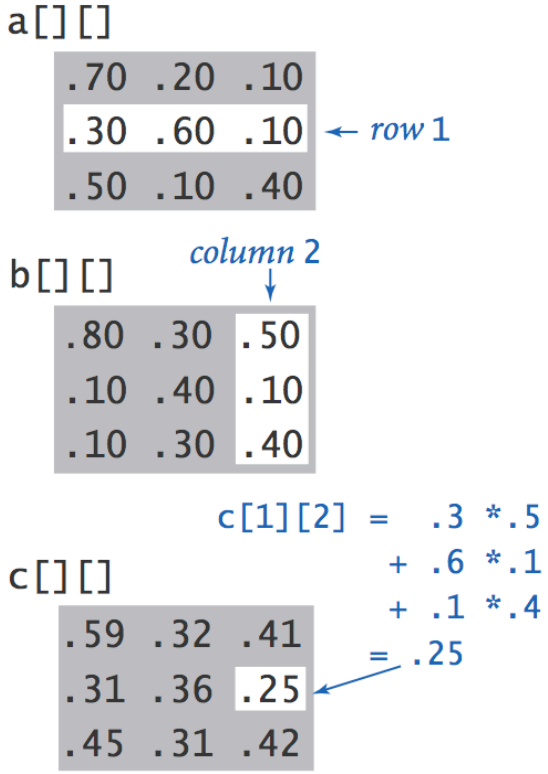
Matrix Multiplication

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[][]` and the j^{th} column of `b[][]`.

all values initialized to 0

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

dot product of row `i` of `a[][]` and column `j` of `b[][]`



Array Challenge

Q. How many scalar multiplications multiply two N-by-N matrices?

A. N B. N^2 C. N^3 D. N^4

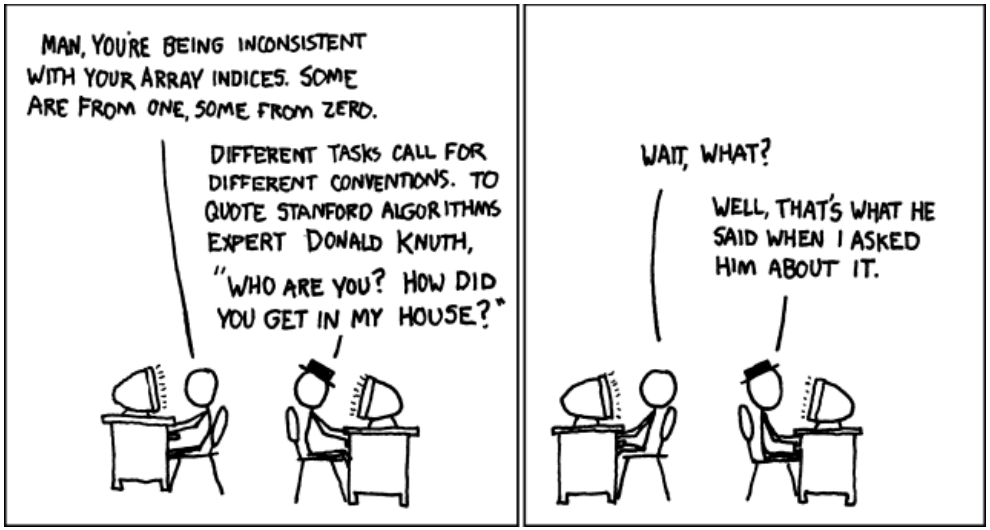
```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

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.

Ahead. Reading in large quantities of data from a file into an array.



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

Exercise 1

Run computational experiments to check that our shuffling code works as advertised. Write a program ShuffleTest that takes command-line arguments M and N, does N shuffles of an array of size M that is initialized with a[i] = i before each shuffle, and prints an M-by-M table such that row i gives the number of times i wound up in position j for all j. All entries in the array should be close to N/M.

*Shuffling algorithm.

```
int M = a.length;
for (int i = 0; i < M; i++) {
   int r = i + (int) (Math.random() * (M-i));
   int t = a[r];
   a[r] = a[i];
   a[i] = t;
}
```

Exercise 2

Alice is throwing a party with N other guests, including Bob. Bob starts a rumor about Alice by telling it to one of the other guests. A person hearing this rumor for the first time will immediately tell it to one other guest, chosen at random from all the people at the party except Alice and the person from whom they heard it. If a person (including Bob) hears the rumor for a second time, he or she will not propagate it further. Write a program to estimate the probability that everyone at the party (except Alice) will hear the rumor before it stops propagating. Also calculate an estimate of the expected number of people to hear the rumor.

1.4 Arrays: Extra Slides

Memory Representation

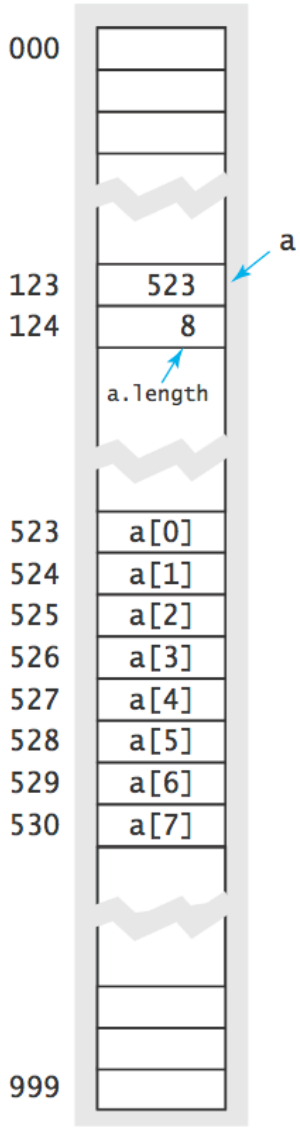
Memory representation. Maps directly to physical hardware.

Consequences.

- Arrays have fixed size.
- Accessing an element by its index is fast.
- Arrays are pointers.

2D array. Array of arrays.

Consequences. Arrays can be ragged.

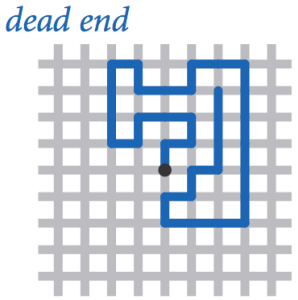


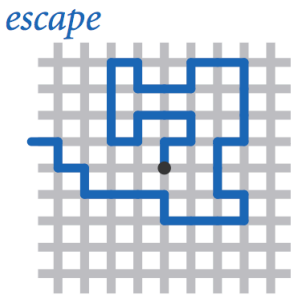
Self-Avoiding Walk

Self-Avoiding Walk

Model.

- N-by-N lattice.
- Start in the middle.
- Randomly move to a neighboring intersection, avoiding all previous intersections.
- Two possible outcomes: dead end and escape.





Applications. Polymers, statistical mechanics, etc.

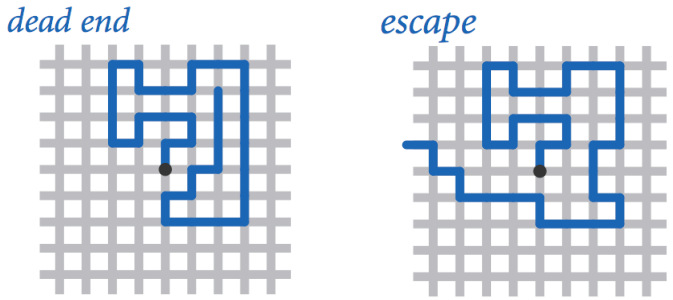
Q. What fraction of time will you escape in an 5-by-5 lattice?

Q. In an N-by-N lattice?

Q. In an N-by-N-by-N lattice?

Self-Avoiding Walk

Skeleton. Before writing any code, write comments to describe what you want your program to do.



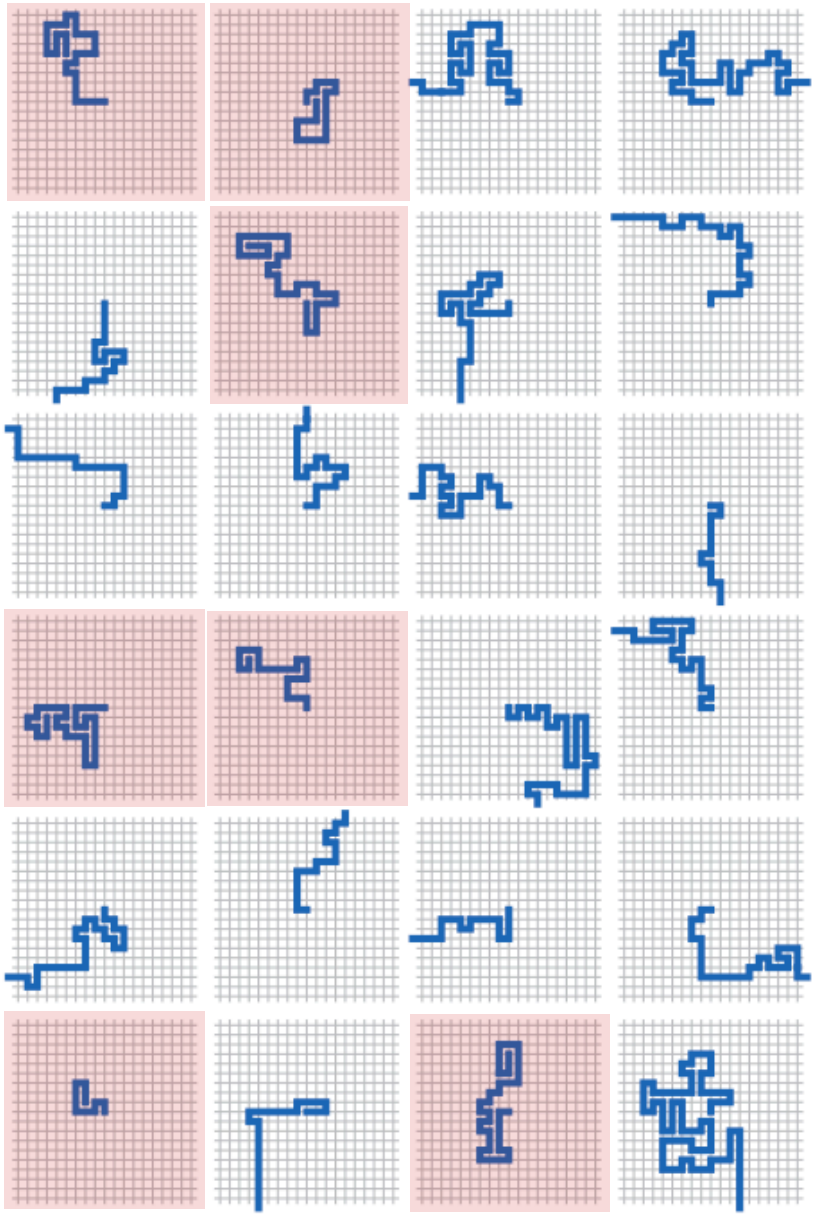

```
public class SelfAvoidingWalk {
   public static void main(String[] args) {
      // Read in lattice size N as command-line argument.
      // Read in number of trials T as command-line argument.

      // Repeat T times:

         // Initialize (x, y) to center of N-by-N grid.

         // Repeat as long as (x, y) stays inside N-by-N grid:
            // Check for dead end and update count.
            // Mark (x, y) as visited.
            // Take a random step, updating (x, y).


         // Print fraction of dead ends.
}
```

how to implement?

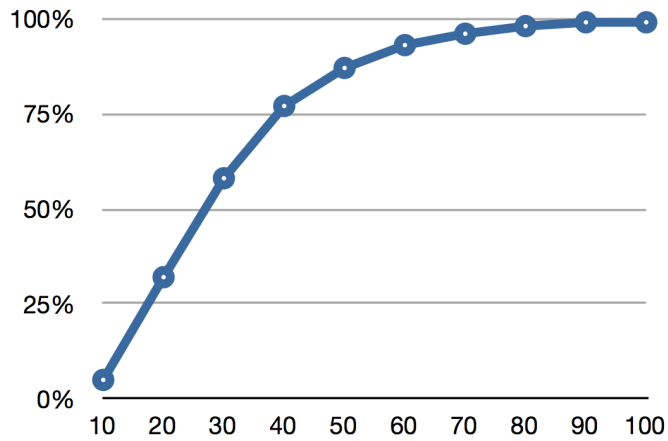
Self-Avoiding Walk: Implementation

```
public class SelfAvoidingWalk {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);      // lattice size
      int T = Integer.parseInt(args[1]);      // number of trials
      int deadEnds = 0;                       // trials resulting in dead end

      for (int t = 0; t < T; t++) {
         boolean[][] a = new boolean[N][N];   // intersections visited
         int x = N/2, y = N/2;                // current position

         while (x > 0 && x < N-1 && y > 0 && y < N-1)  {

            if (a[x-1][y] && a[x+1][y] && a[x][y-1] && a[x][y+1]) {
               deadEnds++;
               break;
            }                                                  dead end

            a[x][y] = true;                   // mark as visited

            double r = Math.random();                  take a random unvisited step
            if      (r < 0.25) { if (!a[x+1][y]) x++; }
            else if (r < 0.50) { if (!a[x-1][y]) x--; }   <-- only take step if site is unoccupied
            else if (r < 0.75) { if (!a[x][y+1]) y++; }
            else if (r < 1.00) { if (!a[x][y-1]) y--; }
         }
      }

      System.out.println(100*deadEnds/T + "% dead ends");
   }
}
```

Visualization of Self-Avoiding Walks



```
% java SelfAvoidingWalks 10 100000
5% dead ends

% java SelfAvoidingWalks 20 100000
32% dead ends

% java SelfAvoidingWalks 30 100000
58% dead ends
...

% java SelfAvoidingWalks 100
100000
99% dead ends
```



Sieve of Eratosthenes

Sieve of Eratosthenes

Prime. An integer > 1 whose only positive factors are 1 and itself.
Ex. 2, 3, 5, 7, 11, 13, 17, 23, ...

Prime counting function. $\pi(N) = \#$ primes $\leq N$.
Ex. $\pi(17) = 7$.

Sieve of Eratosthenes.

- Maintain an array `isPrime[]` to record which integers are prime.
- Repeat for `i=2` to $\sqrt{N}$
  - if `i` is not still marked as prime
    `i` is is not prime since we previously found a factor
  - if `i` is marked as prime
    `i` is prime since it has no smaller factors
    mark all multiples of `i` to be non-prime

Sieve of Eratosthenes

Prime. An integer > 1 whose only positive factors are 1 and itself.
Ex. 2, 3, 5, 7, 11, 13, 17, 23, ...

Prime counting function. $\pi(N) = \#$ primes $\leq N$.
Ex. $\pi(25) = 9$.

i	isPrime																							
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
2	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
3	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T
5	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F
	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F

Trace of java PrimeSieve 25

Sieve of Eratosthenes: Implementation

```
public class PrimeSieve {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);

      // initially assume all integers are prime
      boolean[] isPrime = new boolean[N+1];
      for (int i = 2; i <= N; i++)
         isPrime[i] = true;

      // mark non-primes <= N using Sieve of Eratosthenes
      for (int i = 2; i*i <= N; i++) {
         if (isPrime[i]) {
            for (int j = i; i*j <= N; j++)
               isPrime[i*j] = false;
         }
      }

      // count primes
      int primes = 0;
      for (int i = 2; i <= N; i++)
         if (isPrime[i]) primes++;
      StdOut.println("The number of primes <= " + N + " is " + primes);
   }
}
```