3. Arrays
3. Arrays

- Basic concepts
- Typical array-processing code
- Two-dimensional arrays
Basic building blocks for programming

<table>
<thead>
<tr>
<th>Basic blocks</th>
<th>Ability to store and process huge amounts of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>objects</td>
<td></td>
</tr>
<tr>
<td>functions and modules</td>
<td></td>
</tr>
<tr>
<td>graphics, sound, and image I/O</td>
<td></td>
</tr>
<tr>
<td>arrays</td>
<td></td>
</tr>
<tr>
<td>conditionals and loops</td>
<td></td>
</tr>
<tr>
<td>Math</td>
<td></td>
</tr>
<tr>
<td>text I/O</td>
<td></td>
</tr>
<tr>
<td>primitive data types</td>
<td></td>
</tr>
<tr>
<td>assignment statements</td>
<td></td>
</tr>
</tbody>
</table>

Any program you might want to write
Your first data structure

A data structure is an arrangement of data that enables efficient processing by a program.

An array is an indexed sequence of values of the same type.

Examples.
- 52 playing cards in a deck.
- 100 thousand students in an online class.
- 1 billion pixels in a digital image.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 86 billion neurons in the brain.
- 50 trillion cells in the human body.
- $6.02 \times 10^{23}$ particles in a mole.

Main purpose. Facilitate storage and manipulation of data.
Processing many values of the same type

10 values, without arrays

```java
double a0 = 0.0;
double a1 = 0.0;
double a2 = 0.0;
double a3 = 0.0;
double a4 = 0.0;
double a5 = 0.0;
double a6 = 0.0;
double a7 = 0.0;
double a8 = 0.0;
double a9 = 0.0;
...
da4 = 3.0;
...
da8 = 8.0;
...
double x = a4 + a8;
```

10 values, with an array

```java
double[] a;
a = new double[10];
...
a[4] = 3.0;
...
a[8] = 8.0;
...
double x = a[4] + a[8];
```

1 million values, with an array

```java
double[] a;
a = new double[1000000];
...
a[234567] = 3.0;
...
a[876543] = 8.0;
...
double x = a[234567] + a[876543];
```

tedious and error-prone code

an easy alternative

scales to handle huge amounts of data
Memory representation of an array

An array is an indexed sequence of values of the same type.

A computer's memory is also an indexed sequence of memory locations.
- Each primitive type value occupies a fixed number of locations.
- Array values are stored in contiguous locations.

Critical concepts
- Indices start at 0.
- Given i, the operation of accessing the value \( a[i] \) is extremely efficient.
- The assignment \( b = a \) makes the names \( b \) and \( a \) refer to the same array.

for simplicity in this lecture, think of \( a \) as the memory address of the first location
the actual implementation in Java is just slightly more complicated.

it does not copy the array, as with primitive types
(stay tuned for details)
### Java language support for arrays

#### Basic support

<table>
<thead>
<tr>
<th>Operation</th>
<th>Typical code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare an array</td>
<td>double[] a;</td>
</tr>
<tr>
<td>Create an array of a given length</td>
<td>a = new double[1000];</td>
</tr>
<tr>
<td>Refer to an array entry by index</td>
<td>a[i] = b[j] + c[k];</td>
</tr>
<tr>
<td>Refer to the length of an array</td>
<td>a.length;</td>
</tr>
</tbody>
</table>

#### Initialization options

<table>
<thead>
<tr>
<th>Operation</th>
<th>Typical code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default initialization to 0 for numeric types</td>
<td>a = new double[1000];</td>
</tr>
<tr>
<td>Declare, create and initialize in one statement</td>
<td>double[] a = new double[1000];</td>
</tr>
<tr>
<td>Initialize to literal values</td>
<td>double[] x = { 0.3, 0.6, 0.1 };</td>
</tr>
</tbody>
</table>

BUT cost of creating an array is proportional to its length. No need to use a loop like:

```java
for (int i = 0; i < 1000; i++)
a[i] = 0.0;
```
Copying an array

To copy an array, create a new array, then copy all the values.

```java
double[] b = new double[a.length];
for (int i = 0; i < a.length; i++)
    b[i] = a[i];
```

Important note: The code `b = a` does not copy an array (it makes `b` and `a` refer to the same array).

```java
double[] b = new double[a.length];
b = a;
```
Programming with arrays: typical examples

Access command-line args in system array
```java
int stake = Integer.parseInt(args[0]);
int goal = Integer.parseInt(args[1]);
int trials = Integer.parseInt(args[2]);
```

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

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

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

Find the maximum of array values
```java
double max = a[0];
for (int i = 1; i < N; i++)
    if (a[i] > max) max = a[i];
```

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

For brevity, N is a.length and b.length in all this code.

For brevity, N is a.length and b.length in all this code.

Access command-line args in system array
```java
int stake = Integer.parseInt(args[0]);
int goal = Integer.parseInt(args[1]);
int trials = Integer.parseInt(args[2]);
```

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

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

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

Find the maximum of array values
```java
double max = a[0];
for (int i = 1; i < N; i++)
    if (a[i] > max) max = a[i];
```

Copy to another array
```java
double[] b = new double[N];
for (int i = 0; i < N; i++)
    b[i] = a[i];
```
Pop quiz 1 on arrays

Q. What does the following code print?

```java
public class PQarray1 {
    public static void main(String[] args) {
        int[] a = new int[6];
        int[] b = new int[a.length];

        b = a;
        for (int i = 1; i < b.length; i++)
            b[i] = i;

        for (int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();

        for (int i = 0; i < b.length; i++)
            System.out.print(b[i] + " ");
        System.out.println();
    }
}
```
Pop quiz 1 on arrays

Q. What does the following code print?

```java
public class PQarray1 {
    public static void main(String[] args) {
        int[] a = new int[6];
        int[] b = new int[a.length];
        b = a;
        for (int i = 1; i < b.length; i++)
            b[i] = i;
        for (int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();
        for (int i = 0; i < b.length; i++)
            System.out.print(b[i] + " ");
        System.out.println();
    }
}
```

A. % java PQ4_1
0 1 2 3 4 5
0 1 2 3 4 5

After this, b and a refer to the same array
Programming with arrays: typical bugs

Array index out of bounds

```java
double[] a = new double[10];
for (int i = 1; i <= 10; i++)
    a[i] = Math.random();
```

No a[10] (and a[0] unused)

Uninitialized array

```java
double[] a;
for (int i = 0; i < 9; i++)
    a[i] = Math.random();
```

Never created the array

Undeclared variable

```java
a = new double[10];
for (int i = 0; i < 10; i++)
    a[i] = Math.random();
```

What type of data does a refer to?
Image sources

http://commons.wikimedia.org/wiki/File:CERN_Server_03.jpg
3. Arrays

- Basic concepts
- Examples of array-processing code
- Two-dimensional arrays
**Example of array use: create a deck of cards**

**Define three arrays**
- Ranks.
- Suits.
- Full deck.

```java
String[] suit = {"♣", "♦", "♥", "♠"};
String[] deck = new String[52];

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

Use nested for loops to put all the cards in the deck.

Better style to use `rank.length` and `suit.length` clearer in lecture to use 4 and 13
Example of array use: create a deck of cards

```java
public class Deck {
    public static void main(String[] args) {
        String[] suit = {"♣", "♦", "♥", "♠"};

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

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

% java Deck

```
2♣ 3♣ 4♣ 5♣ 6♣ 7♣ 8♣ 9♣ 10♣ J♣ Q♣ K♣ A♣
2♦ 3♦ 4♦ 5♦ 6♦ 7♦ 8♦ 9♦ 10♦ J♦ Q♦ K♦ A♦
2♥ 3♥ 4♥ 5♥ 6♥ 7♥ 8♥ 9♥ 10♥ J♥ Q♥ K♥ A♥
2♠ 3♠ 4♠ 5♠ 6♠ 7♠ 8♠ 9♠ 10♠ J♠ Q♠ K♠ A♠
%
```
Pop quiz 2 on arrays

Q. What happens if the order of the for loops in Deck is switched?

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

```
for (int i = 0; i < 13; i++)
    for (int j = 0; j < 4; j++)
        deck[i + 13*j] = rank[i] + suit[j];
```
Pop quiz 2 on arrays

Q. What happens if the order of the for loops in Deck is switched?

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

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

A. The array is filled in a different order, but the output is the same.

```
    j
  0 1 2 3
suit ♠ ♦ ♥ ♣

    i
  0 1 2 3 4 5 6 7 8 9 10 11 12
rank 2 3 4 5 6 7 8 9 10 J Q K A
 0 1 2 ... 12 13 14 15 ... 25 26 27 28 ... 38 39 40 41 ... 51
deck 2♣ 3♣ 4♣ ... A♣ 2♦ 3♦ 4♦ ... A♦ 2♥ 3♥ 4♥ ... A♥ 2♠ 3♠ 4♠ ... A♠
```

NOTE: Error on page 92 in 3rd printing of text (see errata list on booksite).
Pop quiz 3 on arrays

Q. Change Deck to put the cards in rank order in the array.

% java Deck
2♣ 2♦ 2♠ 3♣ 3♦ 3♥ 3♠ 4♣ 4♦ 4♥ 4♠ 5♣ 5♦ 5♥ 5♠ 6♣ 6♦ 6♥ 6♠ 7♣ 7♦ 7♥ 7♠ 8♣ 8♦ 8♥ 8♠ 9♣ 9♦ 9♥ 9♠ 10♣ 10♦ 10♥ 10♠ J♣ J♦ J♥ J♠ Q♣ Q♦ Q♥ Q♠ K♣ K♦ K♥ K♠ A♣ A♦ A♥ A♠ %
Pop quiz 3 on arrays

**Q.** Change Deck to put the cards in rank order in the array.

```java
% java Deck
2♣ 2♦ 2♥ 2♠ 3♣ 3♦ 3♥ 3♠ 4♣ 4♦ 4♥ 4♠ 5♣ 5♦ 5♥ 5♠ 6♣ 6♦ 6♥ 6♠ 7♣ 7♦ 7♥ 7♠ 8♣ 8♦ 8♥ 8♠ 9♣ 9♦ 9♥ 9♠
10♣ 10♦ 10♥ 10♠ J♣ J♦ J♥ J♠ Q♣ Q♦ Q♥ Q♠ K♣ K♦ K♥ K♠ A♣ A♦ A♥ A♠
%
```

**A.**

```java
for (int i = 0; i < 13; i++)
    for (int j = 0; j < 4; j++)
        deck[4*i + j] = rank[i] + suit[j];
```
Take a card!  
Any card!

That's my credit card.

Abra kadabra.
Array application: take a card, any card

**Problem:** Print a random sequence of $N$ cards.

**Algorithm**
Take $N$ from the command line and do the following $N$ times
- Calculate a random index $r$ between 0 and 51.
- Print $deck[r]$.

**Implementation:** Add this code instead of printing $deck$ in Deck.

```java
for (int i = 0; i < N; i++)
{
    int r = (int) (Math.random() * 52);
    System.out.println(deck[r]);
}
```

**Note:** Same method is effective for printing a random sequence from any data collection.
public class DrawCards{
    public static void main(String[] args){
      int N = Integer.parseInt(args[0]);

      String[] suit = {"♣", "♦", "♥", "♠"};

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

      for (int i = 0; i < N; i++)
        {  
          int r = (int)(Math.random() * 52);
          System.out.print(deck[r] + " ");
        }
    System.out.println();
    }
}
Array application: shuffle and deal from a deck of cards

Problem: Print $N$ random cards from a deck.

Algorithm: Shuffle the deck, then deal.
- Consider each card index $i$ from 0 to 51.
- Calculate a random index $r$ between $i$ and 51.
- Exchange deck[$i$] with deck[$r$]
- Print the first $N$ cards in the deck.

Implementation

```java
for (int i = 0; i < 52; i++)
{
    int r = i + (int) (Math.random() * (52-i));
    String t = deck[r];
    deck[r] = deck[i];
    deck[i] = t;
}
for (int i = 0; i < N; i++)
    System.out.print(deck[i]);
System.out.println();
```
Array application: shuffle a deck of 10 cards (trace)

```java
for (int i = 0; i < 10; i++)
{
    int r = i + (int) (Math.random() * (10-i));
    String t = deck[r];
    deck[r] = deck[i];
    deck[i] = t;
}
```

Q. Why does this method work?

- Uses only exchanges, so the deck after the shuffle has the same cards as before.
- \(N-i\) equally likely values for \(deck[i]\).
- Therefore \(N \times (N-1) \times (N-1) \ldots \times 2 \times 1 = N!\) equally likely values for \(deck[]\).

Initial order is immaterial.

Note: Same method is effective for randomly rearranging any type of data.
public class DealCards
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);

        String[] suit = { "♣", "♦", "♥", "♠"};

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

        for (int i = 0; i < 52; i++)
        {
            int r = i + (int) (Math.random() * (52-i));
            String t = deck[r];
            deck[r] = deck[i];
            deck[i] = t;
        }

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

**Coupon collector problem**
- \( M \) different types of coupons.
- Collector acquires random coupons, one at a time, each type equally likely.
**Q.** What is the expected number of coupons needed to acquire a full collection?

**Example:** Collect all ranks in a random sequence of cards (\( M = 10 \))

**Sequence**

\[
\begin{align*}
9\spadesuit & \ 5\spadesuit & \ 8\heartsuit & \ 10\diamondsuit & \ 2\clubsuit & \ A\clubsuit & \ 10\heartsuit & \ Q\spadesuit & \ 3\spadesuit & \ 9\heartsuit & \ 5\heartsuit & \ 9\heartsuit & \ 7\diamondsuit & \ 2\diamondsuit & \ 8\clubsuit & \ 6\clubsuit & \ Q\heartsuit & \ K\spadesuit & \ 10\spadesuit & \ A\heartsuit & \ 4\heartsuit & \ J\heartsuit
\end{align*}
\]

**Collection**

\[
\begin{align*}
2 & \ 3 & \ 4 & \ 5 & \ 6 & \ 7 & \ 8 & \ 9 & \ 10 & \ J & \ Q & \ K & \ A \\
2\spadesuit & \ 3\spadesuit & \ 4\spadesuit & \ 5\spadesuit & \ 6\spadesuit & \ 7\spadesuit & \ 8\heartsuit & \ 9\spadesuit & \ 10\spadesuit & \ J\spadesuit & \ Q\spadesuit & \ K\spadesuit & \ A\spadesuit \\
2\clubsuit & \ 5\clubsuit & \ 8\spadesuit & \ 9\heartsuit & \ 10\heartsuit & \ Q\heartsuit & \ A\spadesuit \\
9\spadesuit & \ 10\heartsuit
\end{align*}
\]

22 cards needed to complete collection
Array application: coupon collector

Coupon collector simulation

- Generate random int values between 0 and M−1.
- Count number used to generate each value at least once.

Key to the implementation

- Create a boolean array of length M. (Initially all false by default.)
- When r generated, check the rth value in the array.
  - If true, ignore it (not new).
  - If false, count it as new distinct value (and set rth entry to true)

```java
public class Coupon {
    public static void main(String[] args) {
        int M = Integer.parseInt(args[0]);
        int cards = 0;    // number of cards collected
        int distinct = 0; // number of distinct cards

        boolean[] found = new boolean[M];
        while (distinct < M) {
            int r = (int) (Math.random() * M);
            cards++;
            if (!found[r]) {
                distinct++;
                found[r] = true;
            }
        }
        System.out.println(cards);
    }
}
```

% java Coupon 13
46
% java Coupon 13
22
% java Coupon 13
54
% java Coupon 13
27
Array application: coupon collector (trace for $M = 6$)

```java
boolean[] found = new boolean[M];
while (distinct < M)
{
    int r = (int) (Math.random() * M);
cards++;
    if (!found[r])
    {
        distinct++;
        found[r] = true;
    }
}
```

<table>
<thead>
<tr>
<th>r</th>
<th>found</th>
<th>distinct</th>
<th>cards</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>
Simulation, randomness, and analysis (revisited)

Coupon collector problem
- \( M \) different types of coupons.
- Collector acquires random coupons, one at a time, each type equally likely.

Q. What is the expected number of coupons needed to acquire a full collection?

A. (known via mathematical analysis for centuries) About \( M \ln M + .57721M \).

<table>
<thead>
<tr>
<th>type</th>
<th>( M )</th>
<th>( \text{expected wait} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>playing card suits</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>playing card ranks</td>
<td>13</td>
<td>41</td>
</tr>
<tr>
<td>baseball cards</td>
<td>1200</td>
<td>9201</td>
</tr>
<tr>
<td>Magic™ cards</td>
<td>12534</td>
<td>125508</td>
</tr>
</tbody>
</table>

Remarks
- Computer simulation can help validate mathematical analysis.
- Computer simulation can also validate software behavior.

Example: Is Math.random() simulating randomness?

Pierre-Simon Laplace 1749-1827
Simulation, randomness, and analysis (revisited)

Once simulation is debugged, experimental evidence is easy to obtain.

Gambler's ruin simulation, previous lecture

```java
public class Gambler
{
    public static void main(String[] args)
    {
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        int trials = Integer.parseInt(args[2]);

        int wins = 0;
        for (int i = 0; i < trials; i++)
        {
            int t = stake;
            while (t > 0 && t < goal)
            {
                if (Math.random() < 0.5) t++;
                else t--;
            }
            if (t == goal) wins++;
        }
        System.out.println(wins + " wins of " + trials);
    }
}
```

Analogous code for coupon collector, this lecture

```java
public class CouponCollector
{
    public static void main(String[] args)
    {
        int M = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);
        int cards = 0;
        boolean[] found;

        for (int i = 0; i < trials; i++)
        {
            int distinct = 0;
            found = new boolean[M];
            while (distinct < M)
            {
                int r = (int) (Math.random() * M);
                cards++;
                if (!found[r])
                {
                    distinct++;
                    found[r] = true;
                }
            }
        }
        System.out.println(cards/trials);
    }
}
```
Simulation, randomness, and analysis (revisited)

Coupon collector problem
• $M$ different types of coupons.
• Collector acquires random coupons, one at a time, each type equally likely.
Q. What is the expected number of coupons needed to acquire a full collection?

Predicted by mathematical analysis

<table>
<thead>
<tr>
<th>type</th>
<th>$M$</th>
<th>$M \ln M + 0.57721 M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>playing card suits</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>playing card ranks</td>
<td>13</td>
<td>41</td>
</tr>
<tr>
<td>playing cards</td>
<td>52</td>
<td>236</td>
</tr>
<tr>
<td>baseball cards</td>
<td>1200</td>
<td>9201</td>
</tr>
<tr>
<td>magic cards</td>
<td>12534</td>
<td>125508</td>
</tr>
</tbody>
</table>

Observed by computer simulation

% java CouponCollector 4 1000000 8
% java CouponCollector 13 1000000 41
% java CouponCollector 52 1000000 236
% java CouponCollector 1200 1000000 9201
% java CouponCollector 12534 1000 125508

Hypothesis. Centuries-old analysis is correct and Math.random() simulates randomness.
Image sources

http://www.vis.gr.jp/~nazoya/cgi-bin/catalog/img/CARDSBIC809_red.jpg
http://www.alegriphotos.com/Shuffling_cards_in_casino-photo-deae1081e5ebc6631d6871f8b320b808.html
http://upload.wikimedia.org/wikipedia/commons/b/bf/Pierre-Simon,_marquis_de_Laplace_(1745-1827)_-_Guérin.jpg
3. Arrays

• Basic concepts
• Examples of array-processing code
• Two-dimensional arrays
Two-dimensional arrays

A two-dimensional array is a *doubly-indexed* sequence of values of the same type.

Examples
- Matrices in math calculations.
- Grades for students in an online class.
- Outcomes of scientific experiments.
- Transactions for bank customers.
- Pixels in a digital image.
- Geographic data
- ...

Main purpose. Facilitate storage and manipulation of data.
Java language support for **two-dimensional** arrays (basic support)

<table>
<thead>
<tr>
<th>operation</th>
<th>typical code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declare a two-dimensional array</strong></td>
<td>double[][] a;</td>
</tr>
<tr>
<td><strong>Create a two-dimensional array of a given length</strong></td>
<td>a = new double[1000][1000];</td>
</tr>
<tr>
<td><strong>Refer to an array entry by index</strong></td>
<td>a[i][j] = b[i][j] * c[j][k];</td>
</tr>
<tr>
<td><strong>Refer to the number of rows</strong></td>
<td>a.length;</td>
</tr>
<tr>
<td><strong>Refer to the number of columns</strong></td>
<td>a[i].length;</td>
</tr>
<tr>
<td><strong>Refer to row i</strong></td>
<td>a[i]</td>
</tr>
</tbody>
</table>

```
a[0] [0]  a[0] [1]  a[0] [2]  a[0] [3]  a[0] [4]  a[0] [5]  a[0] [6]  a[0] [7]  a[0] [8]  a[0] [9]
```

*a 3-by-10 array*
## Java language support for **two-dimensional** arrays (initialization)

<table>
<thead>
<tr>
<th>operation</th>
<th>typical code</th>
</tr>
</thead>
</table>
| **Default initialization to 0**  
for numeric types | `a = new double[1000][1000];` |
| **Declare, create and initialize**  
in a single statement | `double[][] a = new double[1000][1000];` |
| **Initialize to literal values** | `double[][] p =
{
    { .92, .02, .02, .02, .02 },
    { .02, .92, .32, .32, .32 },
    { .02, .02, .02, .92, .02 },
    { .92, .02, .02, .02, .02 },
    { .47, .02, .47, .02, .02 },
};` |

No need to use nested loops like:
```java
for (int i = 0; i < 1000; i++)
    for (int j = 0; j < 1000; j++)
        a[i][j] = 0.0;
```

But cost of creating an array is proportional to its size.
Application of arrays: vector and matrix calculations

Mathematical abstraction: vector
Java implementation: 1D array

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

Vector addition

Mathematical abstraction: matrix
Java implementation: 2D array

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 addition

.30 .60 .10 + .50 .10 .40 = .80 .70 .50

.30 .60 .10 + .10 .40 .10 = .40 1.0 .20
.50 .10 .40 + .10 .30 .40 = .60 .40 .80
Application of arrays: vector and matrix calculations

Mathematical abstraction: vector
Java implementation: 1D array

Vector dot product

double sum = 0.0;
for (int i = 0; i < N; i++)
    sum += a[i]*b[i];

<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>0.3</td>
<td>0.5</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>1</td>
<td>0.6</td>
<td>0.1</td>
<td>0.06</td>
<td>0.21</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>0.4</td>
<td>0.04</td>
<td>0.25</td>
</tr>
</tbody>
</table>

end-of-loop trace

Matrix multiplication

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];

<table>
<thead>
<tr>
<th></th>
<th>.70</th>
<th>.20</th>
<th>.10</th>
<th>.80</th>
<th>.30</th>
<th>.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>.30</td>
<td>.60</td>
<td>.10</td>
<td>.30</td>
<td>.30</td>
<td>.50</td>
<td></td>
</tr>
<tr>
<td>.50</td>
<td>.10</td>
<td>.40</td>
<td>.10</td>
<td>.40</td>
<td>.40</td>
<td></td>
</tr>
</tbody>
</table>

=  | .59 | .32 | .41 | .31 | .36 | .25 | .45 | .31 | .42
Pop quiz 4 on arrays

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

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];

1. $N$
2. $N^2$
3. $N^3$
4. $N^4$
Pop quiz 4 on arrays

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

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];

1. $N$
2. $N^2$
3. $N^3$ *Nested for loops: $N \times N \times N$*
4. $N^4$
Self-avoiding random walks

A dog walks around at random in a city, never revisiting any intersection.

Q. Does the dog escape?

Model: a random process in an $N$-by-$N$ lattice
- Start in the middle.
- Move to a random neighboring intersection but do not revisit any intersection.
- Outcome 1 (escape): reach edge of lattice.
- Outcome 2 (dead end): no unvisited neighbors.

Q. What are the chances of reaching a dead end?

Self-avoiding random walks
public class SelfAvoidingWalker
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);
        int deadEnds = 0;
        for (int t = 0; t < trials; t++)
        {
            boolean[][] a = new boolean[N][N];
            int x = N/2, y = N/2;
            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; }
                a[x][y] = true;
                double r = Math.random();
                if       (r < 0.25) { if (!a[x+1][y]) x++; }  
                else if (r < 0.50) { if (!a[x-1][y]) x--; }  
                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/trials + "% dead ends");
    }
}
Simulation, randomness, and analysis (revisited again)

Self-avoiding walk in an $N$-by-$N$ lattice
- Start in the middle.
- Move to a random neighboring intersection (do not revisit any intersection).

Applications
- Model the behavior of solvents and polymers.
- Model the physics of magnetic materials.
- (many other physical phenomena)

Q. What is the probability of reaching a dead end?

A. Nobody knows (despite decades of study).

A. 99+% for $N > 100$ (clear from simulations).

Remark: Computer simulation is often the only effective way to study a scientific phenomenon.
Your first data structure

Arrays: A basic building block in programming
• They enable storage of large amounts of data (values all of the same type).
• With an index, a program can instantly access a given value.
• Efficiency derives from low-level computer hardware organization (stay tuned).

Some applications in this course:

- LFSR
- digital images
- N-body simulation
- digital audio
Image sources


CS.3.C.Arrays.2D
3. Arrays