3. Arrays
3. Arrays

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

- any program you might want to write
- objects
- functions and modules
- graphics, sound, and image I/O
- arrays
- conditionals and loops
- Math
- text I/O
- primitive data types
- assignment statements

Ability to store and process huge amounts of data
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

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;

tedious and error-prone code

10 values, with an array

double[] a;
a = new double[10];
...
a[4] = 3.0;
...
a[8] = 8.0;
...
double x = a[4] + a[8];
an easy alternative

1 million values, with an array

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

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

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

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];
```
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();
    }
}
```

After this, b and a refer to the same array

A. % java PQ4_1
   0 1 2 3 4 5
   0 1 2 3 4 5
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];
```

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

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

Better style to use `rank.length` and `suit.length` clearer in lecture to use 4 and 13

<table>
<thead>
<tr>
<th>rank</th>
<th>suit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 3 4 5 6 7 8 9 10 J Q K A</td>
<td></td>
</tr>
<tr>
<td>♠ ♦ ♥ ♣</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>deck</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ...</td>
</tr>
<tr>
<td>♠ ♣ ♥ ♦</td>
</tr>
</tbody>
</table>
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?

```java
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?

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

NOTE: Error on page 92 in 3rd printing of text (see errata list on book site).
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♠
%
```
Pop quiz 3 on arrays

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

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

```
<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>J</td>
<td>Q</td>
<td>K</td>
<td>A</td>
</tr>
<tr>
<td>j</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>suit</td>
<td>♠</td>
<td>♦</td>
<td>♥</td>
<td>♣</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deck</td>
<td>2♣</td>
<td>2♦</td>
<td>2♥</td>
<td>2♠</td>
<td>3♣</td>
<td>3♦</td>
<td>3♥</td>
<td>3♠</td>
<td>4♣</td>
<td>4♦</td>
<td>4♥</td>
<td>4♠</td>
<td>5♣</td>
</tr>
</tbody>
</table>
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 \( \text{deck}[r] \).

Implementation: Add this code instead of printing \( \text{deck} \) in \text{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.
Array application: random sequence of cards

```java
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();
    }
}
```

Note: Sample is with replacement (same card can appear multiple times).
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();
```

---

Each value between $i$ and 51 equally likely.
Array application: shuffle a deck of 10 cards (trace)

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 \( \text{deck}[i] \).
• Therefore \( N \times (N-1) \times (N-1) \ldots \times 2 \times 1 = N! \) equally likely values for \( \text{deck}[\cdot] \).

Initial order is immaterial.

Note: Same method is effective for randomly rearranging any type of data.
Array application: shuffle and deal from a deck of cards

```java
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();
    }
}
```

Random poker hand:

```
% java DealCards 5
9♣ Q♥ 6♥ 4♦ 2♠
```

Random bridge hand:

```
% java DealCards 13
3♠ 4♥ 10♥ 6♥ 6♦ 2♠ 9♥ 8♠ A♣ 3♥ 9♣ 5♠ Q♥
```
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 ($M =$ )

Sequence

| 9♠ | 5♦ | 8♥ | 10♦ | 2♦ | A♠ | 10♥ | Q♣ | 3♠ | 9♥ | 5♦ | 9♠ | 7♦ | 2♠ | 8♠ | 6♦ | Q♥ | K♣ | 10♥ | A♦ | 4♦ | J♥ |

Collection

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>J</th>
<th>Q</th>
<th>K</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>2♦</td>
<td>3♦</td>
<td>4♦</td>
<td>5♦</td>
<td>6♦</td>
<td>7♦</td>
<td>8♥</td>
<td>9♠</td>
<td>10♦</td>
<td>J♥</td>
<td>Q♣</td>
<td>K♣</td>
<td>A♠</td>
</tr>
<tr>
<td>2♥</td>
<td>5♦</td>
<td>8♠</td>
<td>9♥</td>
<td>10♥</td>
<td>Q♥</td>
<td>A♦</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9♠</td>
<td>10♥</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 $r$th value in the array.
  - If true, ignore it (not new).
  - If false, count it as new distinct value (and set $r$th 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);
    }
}
```
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>2</td>
<td>F</td>
<td>T</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 \( M \ln M + .57721 M \).

<table>
<thead>
<tr>
<th>type</th>
<th>( M )</th>
<th>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?
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 = new boolean[M];

        for (int i = 0; i < trials; i++) {
            int distinct = 0;
            for (int r = 0; r < M; r++) {
                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>$M \ln M + .57721 M$</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 9176
% java CouponCollector 12534 1000 125920

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Typical Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default initialization to 0</strong> for numeric types</td>
<td><code>a = new double[1000][1000];</code></td>
</tr>
<tr>
<td><strong>Declare, create and initialize in a single statement</strong></td>
<td><code>double[][] a = new double[1000][1000];</code></td>
</tr>
</tbody>
</table>
| **Initialize to literal values**               | `double[][] p = {
                { .92, .02, .02, .02, .02 },
                { .02, .92, .32, .32, .32 },
                { .02, .02, .92, .02 },
                { .92, .02, .02, .02 },
                { .47, .02, .47, .02, .02 },
            };
` |
Application of arrays: vector and matrix calculations

**Mathematical abstraction: vector**  
**Java implementation: 1D array**

**Vector addition**

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

![Vector addition example](image)

**Matrix addition**

```java
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 example](image)
Application of arrays: vector and matrix calculations

Mathematical abstraction: vector
Java implementation: 1D array

Vector dot product

```java
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

Mathematical abstraction: matrix
Java implementation: 2D array

Matrix multiplication

```java
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];
```

\[
\begin{array}{ccc}
.30 & .60 & .10 \\
.50 & .10 & .40 \\
\end{array} \times \begin{array}{ccc}
.70 & .20 & .10 \\
.80 & .30 & .50 \\
.30 & .60 & .10 \\
.50 & .10 & .40 \\
.10 & .30 & .40 \\
.10 & .40 & .10 \\
\end{array} = \begin{array}{ccc}
.59 & .32 & .41 \\
.31 & .36 & .25 \\
.45 & .31 & .42 \\
\end{array}
\]
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. $N^3$ \textbf{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
Application of 2D arrays: self-avoiding random walks

```java
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.

Paul Flory
1910-1985
Nobel Prize 1974

Mathematicians and physics researchers cannot solve the problem.

Computational models play an essential role in modern scientific research.
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
- digital audio
- N-body simulation
Image sources

3. Arrays