A peek ahead ... to arrays

We have named variables ...

But each variable holds just one value, and we can’t really manage thousands of them.

Clumsy and Worse ...

Suppose we wanted to compute Muppet change times, excluding the largest two and smallest two values ...  

Possible, but messy.

Suppose we wanted to compute the median (middle value) instead of the average ...

Not even possible.  
Possible, but messy and very slow.

One name, many boxes

int [ ] a = new int [6];
a[3] = 42;
Some details ...

```java
int [ ] a  does not create the boxes;
new array[6] does
```

The variable actually holds a reference (pointer) to the array.
When we create the array with `new`, we have to know how many elements (boxes) to create.

---

Array variables are references

```java
int [ ] a = new int [ 4 ];
```

This is why we can change the items in a method. What we pass to the method is the reference.

---

Initialized arrays

```java
int [ ] daysInMonth =
   new int [ ]  { 0, 31, 28, 31, 30, 31,
            30, 31, 30, 31, 30, 31 };
```

(Size comes from the initialization.)

Why did I make 13 months, and give the first zero days?

---

Arrays and loops: Better together

```java
int [ ] b = new int [ ] { 13, 11, 42, 26, 29 };

int max = -1;  // or some other default
for (int i=0; i < b.length; ++i) {
   if (b[i] > max) {
      max = b[i];
   }
}
```
Better change maker

```java
int [ ] coinValues = new int[ ] { 100, 25, 10, 5, 1 };
String[ ] coinNames = new String[ ]
    { "dollars", "quarters", "dimes", "nickels", "pennies" };

for (int i=0; i < coinValues.length; ++i) {
    int nCoins = change / coinValues[i];
    change = change % coinValues[i];
    System.out.format("%d %s%n", nCoins, coinNames[i]);
}
```

Parallel arrays: coinValues, coinNames

```java
coinValues

<table>
<thead>
<tr>
<th>Dollars</th>
<th>Quarters</th>
<th>Dimes</th>
<th>Nickels</th>
<th>Pennies</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

coinNames

<table>
<thead>
<tr>
<th>Dollars</th>
<th>Quarters</th>
<th>Dimes</th>
<th>Nickels</th>
<th>Pennies</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
```

Why is the new version better?

Less code, for a start
   Fewer chances for mistakes
Less to read
Less to change
And consider modifying it for Euros, Pesos, or Yuan
   Same program logic, different data values

Methods can change arrays

```java
static void swapItems( String [ ] stuff, int i, int k) {
    String tmp = stuff[i];
    stuff[i] = stuff[k];
    stuff[k] = tmp;
}
...
String [ ] greeting = new String [ ]
    { "Yabba", "Dabba", "Doo" };  
swapItems(greeting, 1, 2);
```
Java.util.Arrays

Compare  
Arrays.equals(a,b)

Sort  
Arrays.sort(a)

Search  
Arrays.binarySearch(a, key)

Copy, Fill, ...

If time allows ...

Calculate median time for a Muppet costume change.

Median = “middle value”
median of { 99, 1, 100, 5, 0, 3, 200 } is 5
(three values are smaller, three bigger)

Arrays don’t grow and shrink ... so we create one bigger than we need, and count how much we have used.