Arrays

after an aside on Java API documentation

Aside ... reading API documentation

API = “application programming interface”
“application programming” – that’s us
“interface” – how we can use the libraries

Example: We want to change a string to a character array. How?

The first hit: An example

Convert String to character array

This example shows how to do a String to character array conversion toCharArray().

In the code example below the String is converted to a char array an

```java
public class Main {

    public static void main(String[] args) {
        String str = "HelloWorld";

        char[] charArray = str.toCharArray();
    }
}
```
Refine search to Java 6

These are all API documentation of the JDK libraries ... which is what we want. And from the example we know that we might want toCharArray

Search for toCharArray in Java 6 API ...

The first entry looks like what we want ... class String, with toCharArray()

After some general overview ...

We find lots of methods ...

Including one that looks like what we are after. Click on it.
With full details ...

```

toCharArray
public char[] toCharArray()

Converts this string to a new character array.

Returns:
a newly allocated character array whose length is the length of this string and
whose contents are initialized to contain the character sequence represented by
this string.
```

How do we call it?

```
public char[] toCharArray()

No arguments?? What's with this?

return type: It will produce a value
of type “char []” (array of char)
```

Methods: Static and not static

Compare
```
public static char[] toCharArray(String s)
called like this:
    char [ ] myArray = Array.toCharArray(s)
```
to
```
public char[] toCharArray()
called like this:
    char [ ] myArray = s.toCharArray()
```

Static vs dynamic methods

Static method belongs to the class
We don’t need to create an object to use it

Dynamic methods belong to each object
The object (e.g., the String s) is passed
implicitly as the first argument.
More on this next week ...
One name, many boxes

```
int[] a = new int[6];
a[3] = 42;
```

Some details ...

```
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

```
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.

Matrix (rows and columns)

```
char[][] board = new char[3][3];
```
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?

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);
```

Example: Magic Square

```
  2 7 6
 9 5 1
 4 3 8
```

Example: Magic Square

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>
```

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>
```

```
2 7 6
9 5 1
4 3 8
```
How would you check?

static boolean isMagic( int [][] square ) ...

First: Pseudocode

Then: Java

Get sum of first row ...

int magicSum = 0;
for (int col=0; col < square.length; ++col) {
    magicSum += square[0][col];
}

Other rows the same sum?

for (int row=1; row < square.length; ++row) {
    int rowSum = 0;
    for (int col=0; col < square.length; ++col) {
        rowSum += square[row][col];
    }
    if (rowSum != magicSum) { return false; }

Columns the same sum?

for (int col=0; col < square.length; ++col) {
    int colSum = 0;
    for (int row=0; row < square.length; ++row) {
        colSum += square[row][col];
    }
    if (colSum != magicSum) { return false; }
Diagonals the same sum?

```java
int diagSum = 0;
for (int row=0; row < square.length; ++row) {
    int col = row;  // left to right, downward
    diagSum += square[row][col];
}
if (diagSum != magicSum) { return false; }
```

Diagonal right to left

```java
diagSum = 0;
for (int row=0; row < square.length; ++row) {
    int col = 2 - row;  // diagonal from top right
    diagSum += square[row][col];
}
if (diagSum != magicSum) { return false; }
```

Some applications of matrices ...

Weather prediction, ecosystem simulation
  Each element represents an area
  At each time step, state of element
    changes depending on its neighbors
Medical and research imaging
  3 dimensional grid of “voxels” representing
  a region in the head or body

Indexed Color Images

<table>
<thead>
<tr>
<th></th>
<th>GIF image</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Indexed Color Images

<table>
<thead>
<tr>
<th></th>
<th>GIF image</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Masking: Parallel Arrays

Image processing is matrix processing

Image is matrix (grid) of pixels
Example operations:
- Composite (blend) images
- Blur image, or sharpen edges
- Scale image

How would you do these?
(just strategy or very rough pseudocode)

Blur
For each pixel in blurred image
Value is weighted average of surrounding pixels in old image

Unsharp Mask (edge sharpening)
1. Create a blurred image
2. Subtract blurred image from original (called “high pass filter”)
3. Add high-pass filtered image to original

Each step creates a new matrix of pixels from one or two other matrices, looping through all the pixels
Class exercise: String scrambler

Input: A word on the command line
Output: Scrambled word:
   Same characters, random order

Pseudocode first; then code if time allows

Scrambler Approaches

1\textsuperscript{st} approach suggested by students in class:
   Start with all letters in original word, no letters in new copy.
   While some left in original: Randomly choose one to move to copy. (Move last letter of original into the gap and shorten original.)

2\textsuperscript{nd} approach suggested:
   For each letter in original, choose a random position to swap it with.
   Either one is good. My sample in next slides is like 2\textsuperscript{nd}.

Scrambler method ...

```java
public static String scramble(String s) {
    Random rand = new Random();
    char[] schars = s.toCharArray();
    for (int i = 0; i < schars.length; ++i) {
        int swapPos = rand.nextInt(schars.length);
        swap(schars, i, swapPos);
    }
    return new String(schars);
}
```

Swap array elements

```java
static void swap(char[] letters, int i, int j) {
    char temp = letters[i];
    letters[i] = letters[j];
    letters[j] = temp;
}
```