Previous Class

• What did we discuss?
Today

• Announcements
  • Assignment 2 : Due Friday, Oct 2 @ 11:55 PM
    http://comp110.com/assignments/the-worried-wizard

• Midterm on Thu, Oct 8
  • in class, no computers

• Study guide
  http://comp110.com/midterm-study-guide

• Arrays

Let’s talk Dorms
Proposal: Krzyzewski Dorm

Mike Krzyzewski will donate a lot of money to UNC iff:
In Coach K Dorm, the rooms aren’t numbered, they’re named.

What benefits do room numbers provide?

What are the benefits of a Dorm Name + Number Addressing Scheme?

Everett Dorm:

- Naming is hard. Numbering is easy.
  Learning and remembering names even harder.

- Locate rooms quickly and predictably.

- Easier to manage A LOT of rooms.
  “Ok, I’ll prep rooms 1-150, you prep 151-300.”
Arrays are like **Dorms for Data**

Arrays provide uniform housing for *many* values.

1. Each “room” in an array is called an **Element**
2. An element stores a single value. *No roommates!*
3. All elements in an array are of the same type
4. Arrays cannot be resized after construction
Elements are addressed by **Name and Index**

1. Notation: `arrayName[index]`, i.e. `arrayName[9]`
2. **Indexing starts at [0]** (not [1])
   (Beware: Off-by-one bugs can be stingers 🐝.)

Questions?

Where have we seen `[]`s?
Let’s demystify some magic

```java
public static void main(String[] args)
```

- When you see [brackets] there are arrays
- This is an array of strings available to our program!
- What’s in it?
  - Arguments passed in when we run our program
  - `> java cli.CommandLineDemo foo bar baz`  

Recall Expressions:

**Expressions**

- **Expression?**
  - An **expression** can be a variable, a value, or a combination made up of variables, values and operators
  - An expression has a **value**
  - **Arithmetic expression**: a combination of numbers with a number value
    - 10, `taxRate/100, (cost + tax) * discount`
**The Array Index is an Expression!!!**

This is a major coup.

🎉🎉🎉🎉🎉

### Accessing Elements with an Index

index is any integer expression.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Element</td>
<td>arrayName[index]</td>
<td>scores[0]</td>
</tr>
</tbody>
</table>

##### Number

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrayName[0]</td>
<td></td>
</tr>
</tbody>
</table>

##### Integer

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i = 0;</td>
<td></td>
</tr>
<tr>
<td>arrayName[i]</td>
<td></td>
</tr>
</tbody>
</table>

##### Arithmetic

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i = 0;</td>
<td></td>
</tr>
<tr>
<td>arrayName[i+1]</td>
<td></td>
</tr>
</tbody>
</table>

#### Looping with an Integer

```java
for(int i = 0; i < arrayName.length; i++){
    System.out.println(arrayName[i]);
}
```
## Finding # of Elements

`.length` property is number of elements in array.

Use “< a.length” in for loop termination test.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Elements</td>
<td>arrayName.length</td>
<td>scores.length</td>
</tr>
</tbody>
</table>
How do we make our own arrays?

Declare an Array
int[] yards;

Construct an Array
yards = new int[3];

Assign Elements
yards[0] = 10;
yards[1] = 20;
yards[2] = 30;

How do we average yards?

Java Array Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Elements</td>
<td>arrayName.length</td>
<td>scores.length</td>
</tr>
<tr>
<td>Read Element</td>
<td>arrayName[index]</td>
<td>scores[0]</td>
</tr>
<tr>
<td>Declare</td>
<td>type[] arrayName;</td>
<td>int[] scores;</td>
</tr>
<tr>
<td>Construct</td>
<td>arrayName = new type[size];</td>
<td>scores = new int[3];</td>
</tr>
<tr>
<td>Assign Element</td>
<td>arrayName[index] = expression;</td>
<td>scores[0] = 12;</td>
</tr>
<tr>
<td>Initialize</td>
<td>type[] arrayName = {elements};</td>
<td>int[] scores = {12,0,1};</td>
</tr>
<tr>
<td>(Just a shortcut.)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Key Concepts

1) An array is a uniform structure housing many elements.

2) Elements are addressed via name and index.

3) **Index is an expression!!!**

Arrays

- To think about arrays, let’s think about loops first
- Why do we need loops?
  - Because we want to repeat things without writing them again and again
Average Score without Loops

• Assuming that we only need 5 basketball scores for averaging...

```java
int score1 = keyboard.nextInt();
int score2 = keyboard.nextInt();
int score3 = keyboard.nextInt();
int score4 = keyboard.nextInt();
int score5 = keyboard.nextInt();

double average = (double) (score1 + score2 + score3 + score4 + score5) / 5.0;
```

Average Score with Loops

• Assuming that we only need 5 scores

```java
for (int i = 0; i < 5; i++)
    scoreSum += keyboard.nextInt();

double average = (double) scoreSum / 5.0;
```
What if we really need to save them

• If we really need to save these scores, loop won’t help you
• Think about this problem
  – Print out if a score is above/below average
  – We have to calculate average first, then decide if a score is above/below average
  – Therefore we must save all these scores, and compare them to the average in the end

Comparing All Scores and the Average

```java
System.out.println("Enter 5 basketball scores:");
Scanner keyboard = new Scanner(System.in);
int score1 = keyboard.nextInt();
int score2 = keyboard.nextInt();
int score3 = keyboard.nextInt();
int score4 = keyboard.nextInt();
int score5 = keyboard.nextInt();
double average = (double) (score1 + score2 + score3 + score4 + score5) / 5.0;
System.out.println("Average score: "+average);

// repeat this for each of the 5 scores
if (score1 > average)
    System.out.println(score1 + ": above average");
else if (score1 < average)
    System.out.println(score1 + ": below average");
else
    System.out.println(score1 + ": equal to the average");

// if score2...score3...score4...
```
If we have more Scores...

- Think about 80 scores...
  - Declare 80 variables
  - Check them 80 times
- This is illogical!
- There must be an easier way!
  - What about things like: Score₁, Score₂, ..., Scoreₙ

Arrays

- `int[] scores = new int[5];`
- This is like declaring 5 strangely named variables of type int:
  - scores[0]
  - scores[1]
  - scores[2]
  - scores[3]
  - scores[4]
- Especially, you can use `score[i]` to locate a single one
Arrays

- An **array** is a collection of items of the same type
- Like a list of different variables, but with a nice, compact way to name them
- A special kind of object in Java
- **Loops repeat things temporally; arrays repeat things spatially**

Comparing Scores/Averages w/ Arrays

```java
System.out.println("Enter 5 basketball scores:");
Scanner keyboard = new Scanner(System.in);
int[] scores = new int[5];
int scoreSum = 0;
for (int i = 0; i < 5; i++) {
    scores[i] = keyboard.nextInt();
    scoreSum += scores[i];
}
double average = (double) scoreSum / 5;
System.out.println("Average score: " + average);
for (int i = 0; i < 5; i++) {
    if (scores[i] > average)
        System.out.println(scores[i] + ": above average");
    else if (scores[i] < average)
        System.out.println(scores[i] + ": below average");
    else
        System.out.println(scores[i] + ": equal to the average");
}
```
Index

- Variables such as scores[0] and scores[1] that have an integer expression in square brackets are known as:
  - *indexed variables, subscripted variables, array elements*, or simply *elements*

- An *index* or *subscript* is an integer expression inside the square brackets that indicates an array element
  - ArrayName[index]

Where have we seen the word index before?

- String’s indexOf() method
  - str.indexOf('e') == 6;
  - str.charAt(6) == ‘e’;
Index

• **Index numbers start with 0.** They do NOT start with 1 or any other number.
  – Not like counters in loops, you can’t change the range of indices

• The reason is that the array name represents a memory address, and the i\textsuperscript{th} element can be accessed by the address plus i

Array and Index

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>62</td>
<td>51</td>
<td>88</td>
<td>70</td>
<td>74</td>
</tr>
<tr>
<td>m address</td>
<td>25131</td>
<td>25132</td>
<td>25133</td>
<td>25134</td>
<td>25135</td>
</tr>
</tbody>
</table>

• In history, computer scientists argued a lot on this
  • “Should array indices start at 0 or 1? My compromise of 0.5 was rejected without, I thought, proper consideration.” – Stan Kelly-Bootle
Access Elements with Indices

• The number inside square brackets can be any integer expression
  – An integer: scores[3]
  – Variable of type int: scores[index]
  – Expression that evaluates to int: scores[index*3]

• Can use elements just like any other variables:
  – scores[3] = 68;
  – System.out.println(scores[1]);

Indices and For-Loops

• In programming, a for-loop usually starts with counter i = 0. There is a reason

```
for (int i = 0; i < 5; i++) {
    scores[i] = keyboard.nextInt();
    scoreSum += scores[i];
}
```
Creating an Array

- Array is a special class and we create its objects
  - Syntax for creating an array:
    - `Base_Type[] Array_Name = new Base_Type[Length];`
  - Example:
    - `int[] pressure = new int[100];`
  - Alternatively:
    - `int[] pressure;`
    - `pressure = new int[100];`

Do not be OUT OF BOUNDS!

- Indices MUST be in bounds
  - `double[] entries = new double[5]; // from [0] to [4]`
  - `entries[5] = 3.7; // ERROR! Index out of bounds`
- Your code will compile if you are using an index that is out of bounds, but it will give you a run-time error!
Initializing Arrays

• You can initialize arrays when you declare them
  – `int[] scores = { 68, 97, 102 };`
• Equivalent to
  – `int[] scores = new int[3];`
  – `scores[0] = 68;`
  – `scores[1] = 97;`
  – `scores[2] = 102;`
• Or, you can use for-loop
  – When in doubt, for-loop!

Joke

• Q: Why did the programmer quit his job?
• A: Because he didn't get arrays.
  
  Hint: A raise ;-)
Next class

• More on arrays