Arrays

- From last time:
  - getGrade poll
  - references for other important control structures topics
- Arrays
  - Ex: count scores
  - Example of array uses
  - Random access
  - Syntax
  - Return to counting scores example
  - Arrays of objects
  - Partially filled array
- Introduction to the Java **ArrayList** class

Announcements

- Reminders:
  - PA1 due on Wed 2/10
  - Lab 3 has advanced prep: see lab for details
- Students that need to see me after class today:
  - missed the first lecture,
  - not officially enrolled
  - who have no previous programming experience

From last time…

- Finish multi-way test, **`getGrade`** example . . .

Other important control structures topics in the textbook

- The dangling-else problem: Common Error 5.3
- DeMorgan's laws: Special Topic 5.7
- Hand-tracing: Prog. tip 5.5 and Section 6.2
- Processing sentinel values: Section 6.5, Special topic 6.3
  - Do **`NapCalc.java`** example on your own (starter code and solutions in **`02-02`** lecture dir):
    - Add code to error-check for non-negative input with a **`loop`**.
    - Avoid **`break`** or **`continue`** constructs (they are error prone)
- Another example of multi-way test: Do **`CommandProcessor.java`** example (starter code in **`02-02`** lecture dur)

POLL: unit-test review question…

- Which of the options listed are desirable features of a test driver, such as **`StudentTester`**

- In-class: `pollEv.com/cbono`

Asynchronous participation: Link to Unit-Test poll

Consider this problem…

- read a bunch of student scores in the range 0-10 and determine how many people got each score…

- Some code to do this in ….

`ScoreCountsHard.java`

What arrays are for

- Can store a *collection* of items of the same type using one variable.
- For example:
  - points in an n-sided polygon
  - times of all runners at a track meet
  - distinct words from a story and their frequencies
  - student scores
  - all employees in a department
- also get *random access* . . .

Random access

Examples:

- can go right to track 3 on a CD
- can change individual pixels on a computer monitor
- can access the score for student #4 as fast as student #23
- can solve counting scores problem (store a bunch of counts)

Array syntax

```
int[] temps;
```
array reference

```
temps = new int[10];
```
create array object

valid indices are 0 through 9

```
int aTemp = temps[3];
```
access an array elmt

```
temps[3] = 59;
```
change value of array element

```
int temp2 = temps[10];
```
run-time error

```
int len = temps.length;     // 10
```

`java.util.Arrays` class has useful static array methods.

Using a variable to index an array

```
int aNum = temps[i];
```

- What's a safer way to write this code?

Accessing an array sequentially

Let's print all the values in **`temps`** …

Return to counting scores

- Now we're ready to write better code to solve the counting scores problem:
- read a bunch of student scores and determine how many people got each score…

`ScoreCounts.java`

Arrays of objects

```
String[] names= new String[10];
```

create array of 10 String references

```
int len = names[0].length();
```
run-time error

```
names[0] = "Suzy";
```
now refers to a string object

```
String name = names[10];
```
run-time error

```
len = names[0].length();
```
ok

Array elements have default initialization

- **`new Foo[10]`** all initialized to **`null`**
- **`new int[10]`** all initialized to **`0`**
- **`new boolean[10]`** all initialized to **`false`**
- Reminder:
  - Like instance variables
  - In contrast locals are **not** initialized by default

POLL: Arrays of objects

- The following code goes with the poll:

```
Rectangle[] rectArr = new Rectangle[10];
rectArr[2] = new Rectangle();
```

Hint: it may be helpful to draw a box-and-pointer diagram

Asynchronous participation: Link to Array of Objects poll

Review: applications where we use random access

Characteristics:

- Uses random-access
- Array size known ahead of time and doesn't change

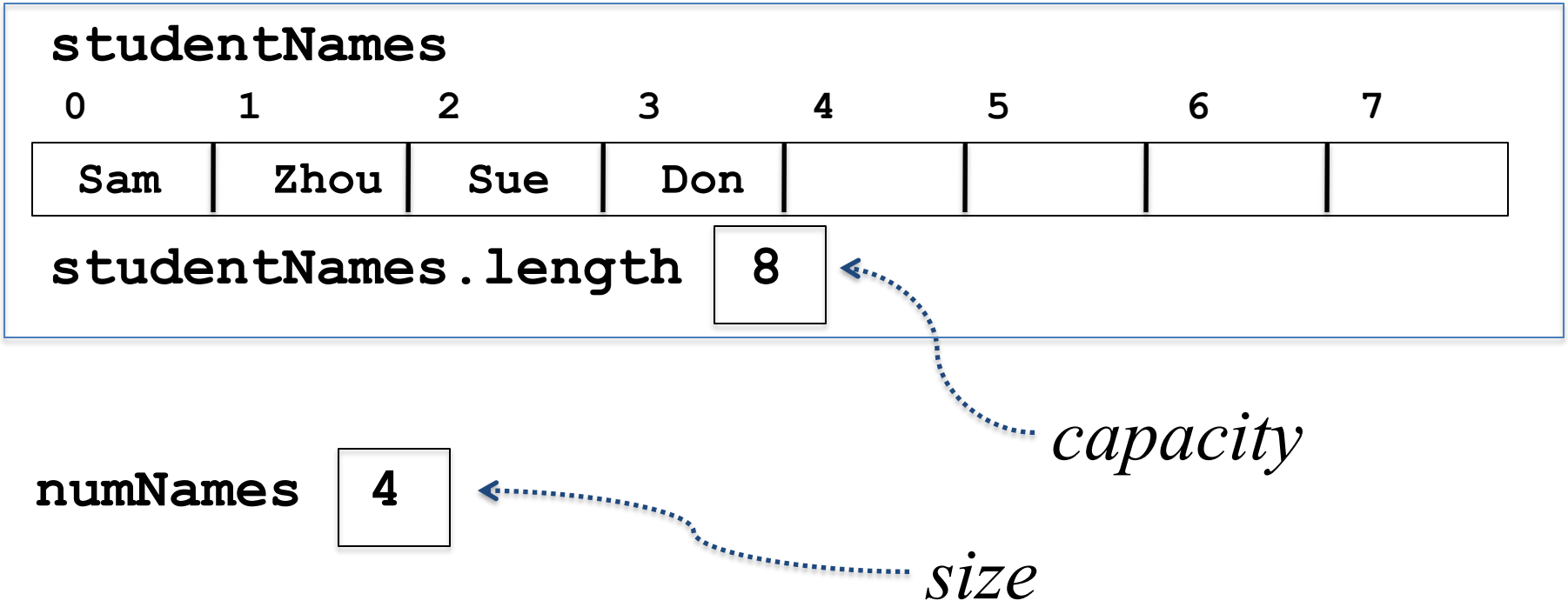
Ex: count how many people got each score (histogram)

Other array applications

- Ex: store data about all students in the class
- Characteristics…
  - Don't know how many students there will be ahead of time
  - Students may add or drop
  - Uses mostly sequential access

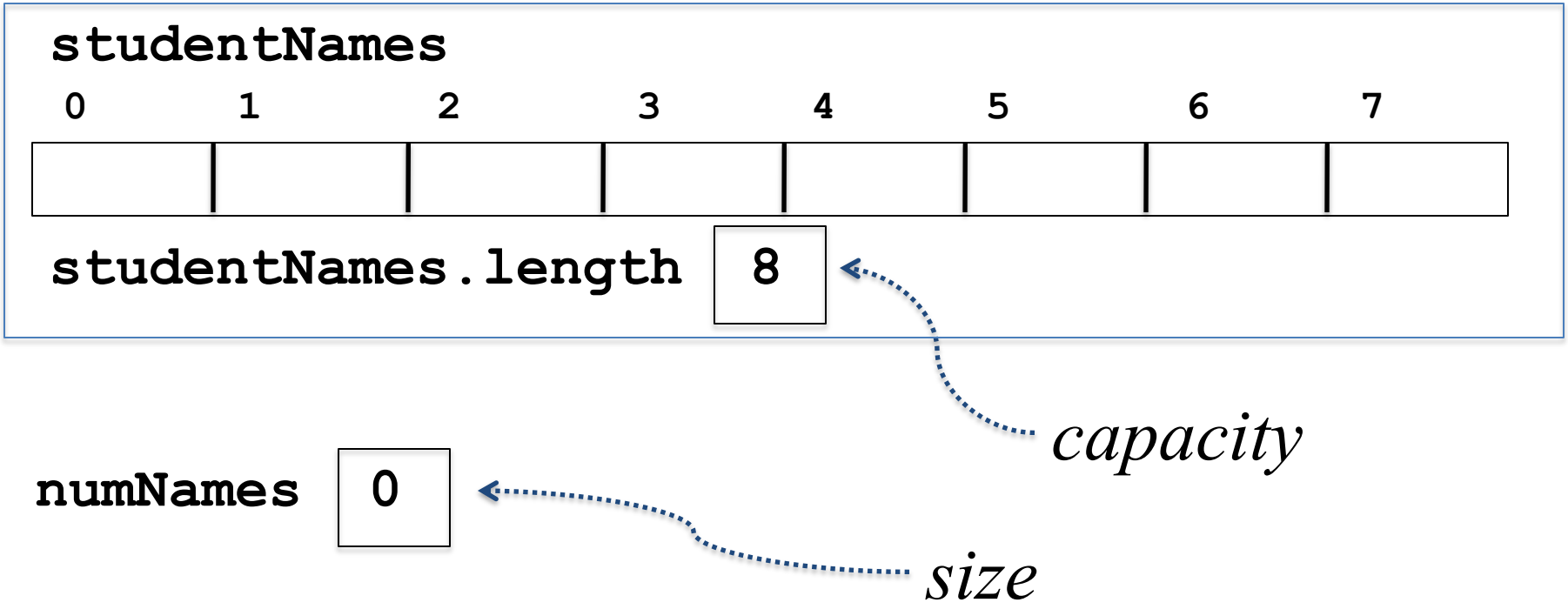
- Use a partially filled array

Ex: partially filled array of student names



code to add a new student to the end:

Empty partially filled array of student names



example initialization:

```
String[] studentNames = new String[8];
int numNames = 0;
```

Difficulties of partially filled array

- have to guess necessary capacity ahead of time
- have to keep two variables in sync: **`numNames`** and **`studentNames`**
- What if we run out of space?
  - have to allocate a bigger array
  - copy all the elements from smaller array to bigger array
  - **`Arrays.copyOf`** (discussed in section 7.3.9) can help with this
- Common use of arrays, so …

`ArrayList` class

- Hides the code to take care of messy details of partially-filled array:
- Keeps track of how full array is:

  `arrList.size()`
- Makes array bigger as necessary:

  `arrList.add("Zhou");`

  adds Joe to the *end* of the partially-filled array
- Accessing individual elements by index still uses random access (fast): **`get`**, **`set`**