`ArrayList`; **`Names`** example

- Review: array syntax
- Partially filled array
- ArrayList (continued)
- Example: **`Names`** class
- practice with
  - coding array algorithms
  - implementing classes
  - and using good development techniques
- incremental development
- for **`lookup`**:
  - design test cases first
  - implement code
    - code refactoring
  - test code
- continue next time

Announcements

- PA1 due Wednesday night
- this week's lab: programming with ArrayList
- Later this week:
  - PA2 will be published
  - Sample MT 1 exams will be available

Review Array syntax

`int[] intArr;` array reference only

`intArr = new int[100];` create array object

valid indices are?

`int val= intArr[10];` access an array elmt

its value is?

`intArr[10] = 59;` change value of array element

`int val2 = intArr [100];` *what does this do?*

complete a loop to add 10 to all the elements in the array:

```
for (int i = 0;                                    ; i++) {


}
```

Review: applications where we use random access

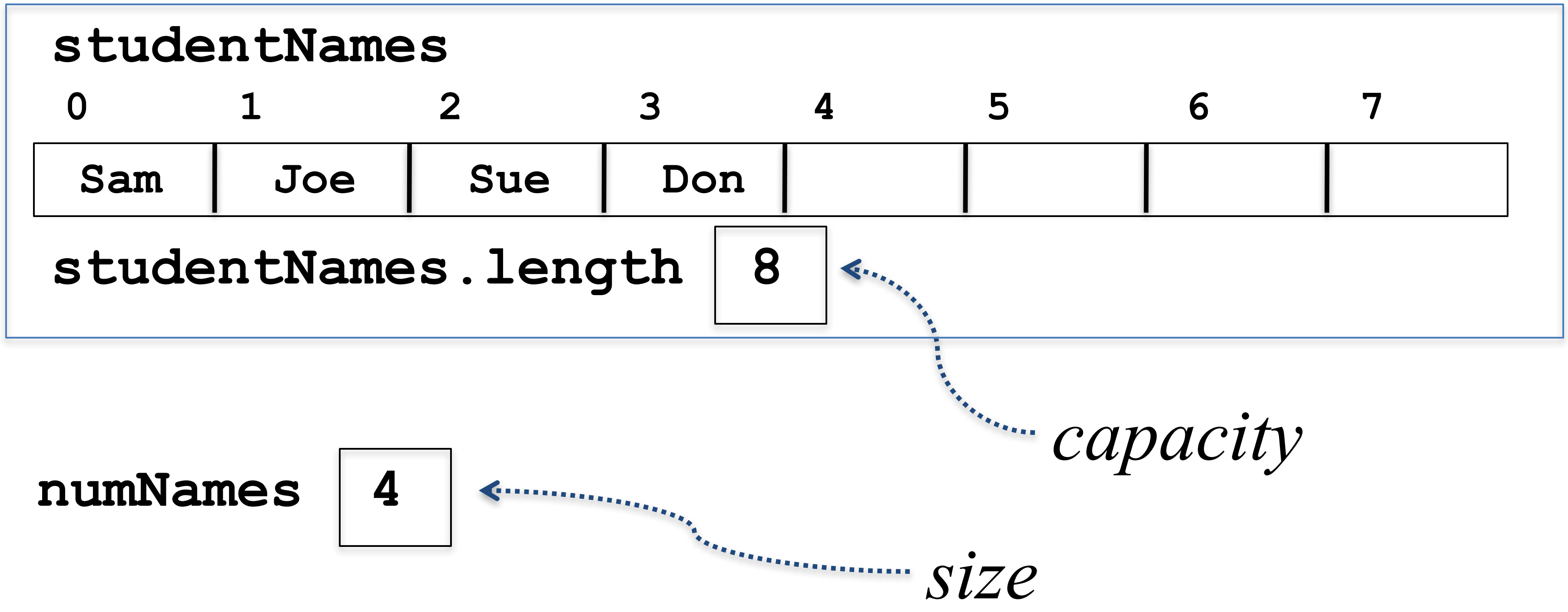
- Ex: count how many people got each score (histogram)
- Use random-access
- Array size known ahead of time and doesn’t change

Partially filled array

- 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

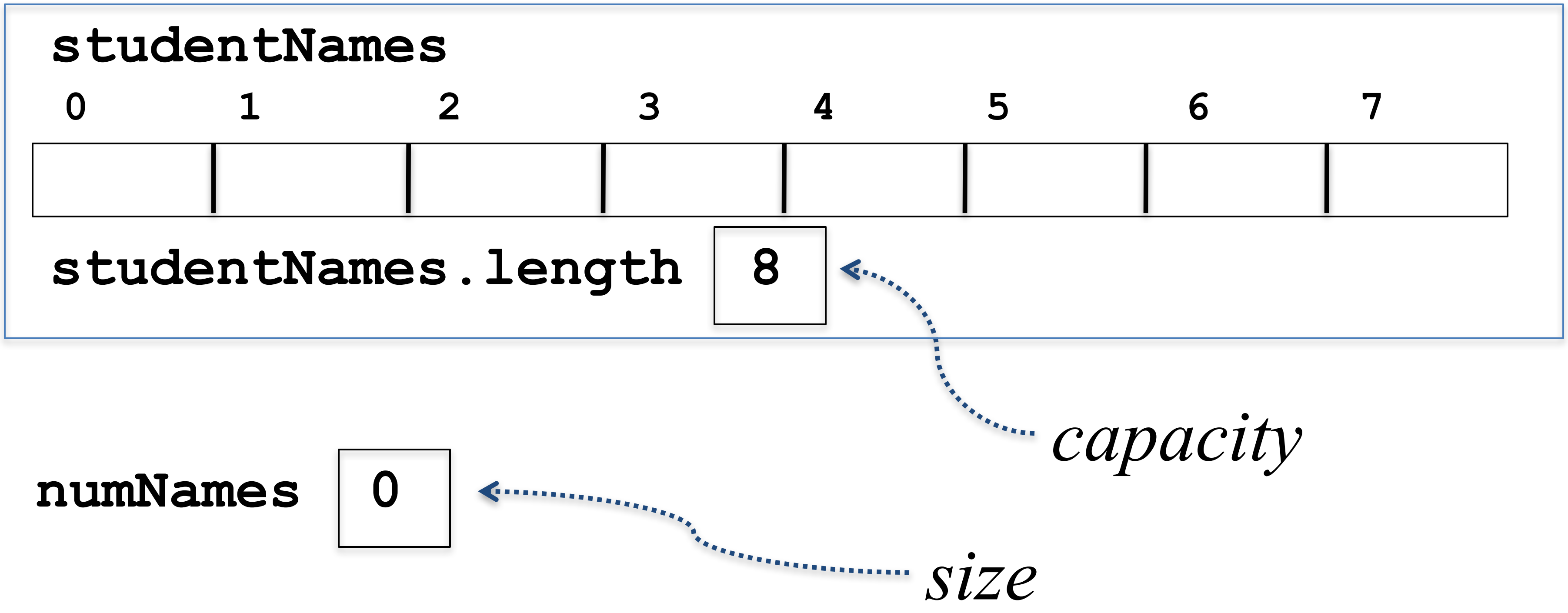
- Use a partially filled array

Ex: partially filled array of student names



add a new name:

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
  - (code example in section 7.3.9 – note: **`copyOf`** *not* available in Java 1.5)
- 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("Joe");`

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

`ArrayList` basic syntax

```
ArrayList<String> names =
                  new ArrayList<String>();
```

create empty arraylist

```
int len = names.size();      // 0
```

```
names.add("Joe");
```
adds a new name to end of list

```
int len = names.size();      // 1
```

```
String name = names.get(0);
```
like **`names[0]`**

```
names.set(0, "Suzy");
```
like **`names[0]=`**

```
String name2 = names.get(1);
```
run-time error

Print out elements of **`ArrayList`**

```
public static void printNames(
                         ArrayList<String> names) {
```

`ArrayList` of ints

- With generics, must use a ***class*** as type parameter:

```
ArrayList<Integer> nums =
              new ArrayList<Integer>();
```

create empty arraylist

```
nums.add(3);
nums.add(17);
nums.add(2);
int n = nums.get(1);
```

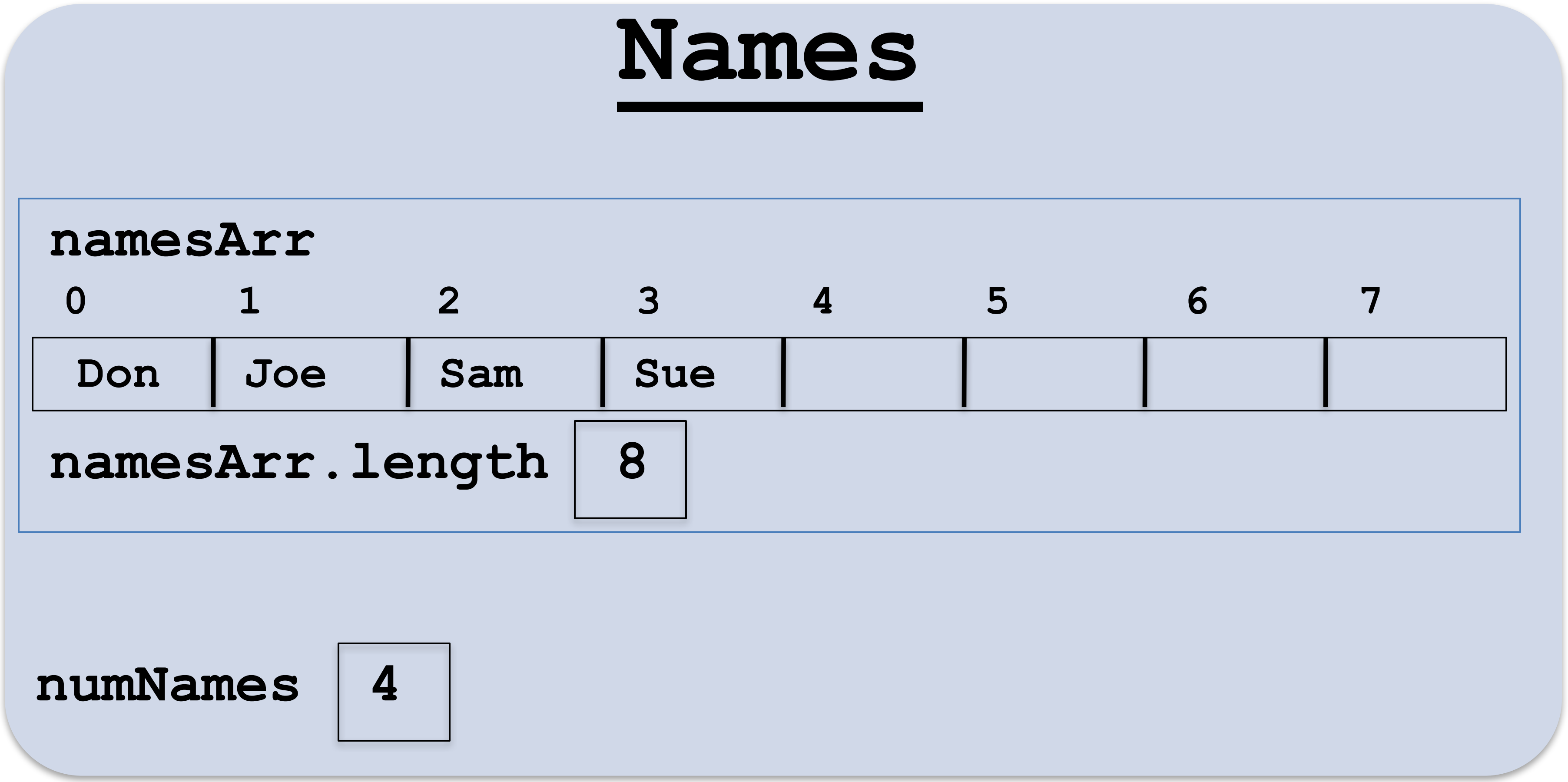
- Uses auto-boxing

`Names` class

- Stores a list of unique names in alphabetical order.
- Allows look-up, insert, and removal of elements in the list.
- Uses partially-filled array representation

- **`Names.java`** has a partial implementation
- **`MinNamesTester.java`** is a program to test that subset.

Names respresentation



Lookup test cases

- Returns true iff **`target`** is present in names

`namesArr`

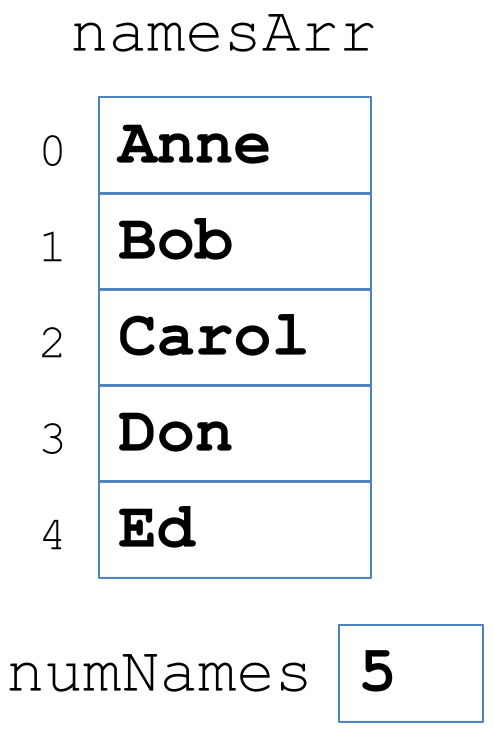
0	**`Anne`**
1	**`Bob`**
2	**`Carol`**
3	**`Don`**
4	**`Ed`**

`numNames` **`5`**

<u>Test cases</u>

Lookup code notes

- Returns true iff **`target`** is present in names


namesArr
0
Anne
1
Bob
2
Carol
3
Don
4
Ed
numNames
5