java.util.Collection ArrayList, HashSet levels of abstraction

https://201wo.to

https://canvas.duke.edu/courses/70546

Owen Astrachan and Alex Steiger

ola@duke.edu, alexander.steiger@duke.edu

D is for …

- Debugging
  - A key skill in making your programs ***correct***, your program will ***run*** (correctly?) without this skill
- Database
  - From SQL to NO-SQL to …
  - From data to information

What's due "soon" in 201

- P0 is due Jan 27, but that means …
  - Early bonus engagement points
  - 48 hours with no penalty
  - -10% day with weekend as one day

- APTs are due Thursday (this week and next)
  - APT-2 out this evening on PrairieLearn
  - Always one late day allowed, no later submissions

Plan for the day

- Java classes: array, ArrayList, HashSet
  - Collection classes in more detail
- Levels of abstraction and measuring efficiency
  - How do Collection classes "work"?
- Project P0: Person201, some details

APT Replay: Starter

- Avoid duplicate entries in an array
  - java.util.ArrayList, holds Objects only, no primitives
  - String? 😊, but int 😢

prints: 5 and 3

```
5       String[] a = {"ant", "bat", "cat", "ant", "bat"};
6       ArrayList<String> list = new ArrayList<>();
7       for(String s : a) {
8           if (! list.contains(s)) list.add(s);
9       }
10      System.out.printf("%d and %d\n",a.length, list.size());
```

Another data structure: HashSet

- `HashSet<String>` or `ArrayList<String>`
  - both support `.add`, `.size`, `.contains`, and more
  - HashSet ***doesn't*** store duplicates, no need to check
- Access via import java.util.*; or java.util.HashSet;

prints: 5 and 3

```
12      HashSet<String> set = new HashSet<>();
13      for(String s : a) set.add(s);
14      System.out.printf("%d and %d\n",a.length, set.size());
```

Two Collection classes (so far) in 201

- ArrayList – grows, *iterable*, *indexable*
  - `.contains(elt)` may look at every element: linear
- HashSet – grows, iterable, ***not*** indexable
  - `.contains(elt)` is *constant time*
    - 10, 100, 1000, or a million elements? Same time

- Both support `.clear, .addAll, .remove`
  - And many more methods

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/Collection.html

Creating java.util.Collection objects

- `Arrays.asList(..)` returns `List<String>`
  - immutable list, convenience, stores values

```
16   String[] arr = {"a","b","c"};
17   List<String> alist = Arrays.asList(arr);
18   ArrayList<String> aa = new ArrayList<>(alist);
19   aa.addAll(alist);
20   HashSet<String> hset = new HashSet<>(alist);
21   hset.addAll(alist);
22   System.out.printf("%d and %d\n",aa.size(), hset.size());
```

prints: 6 and 3

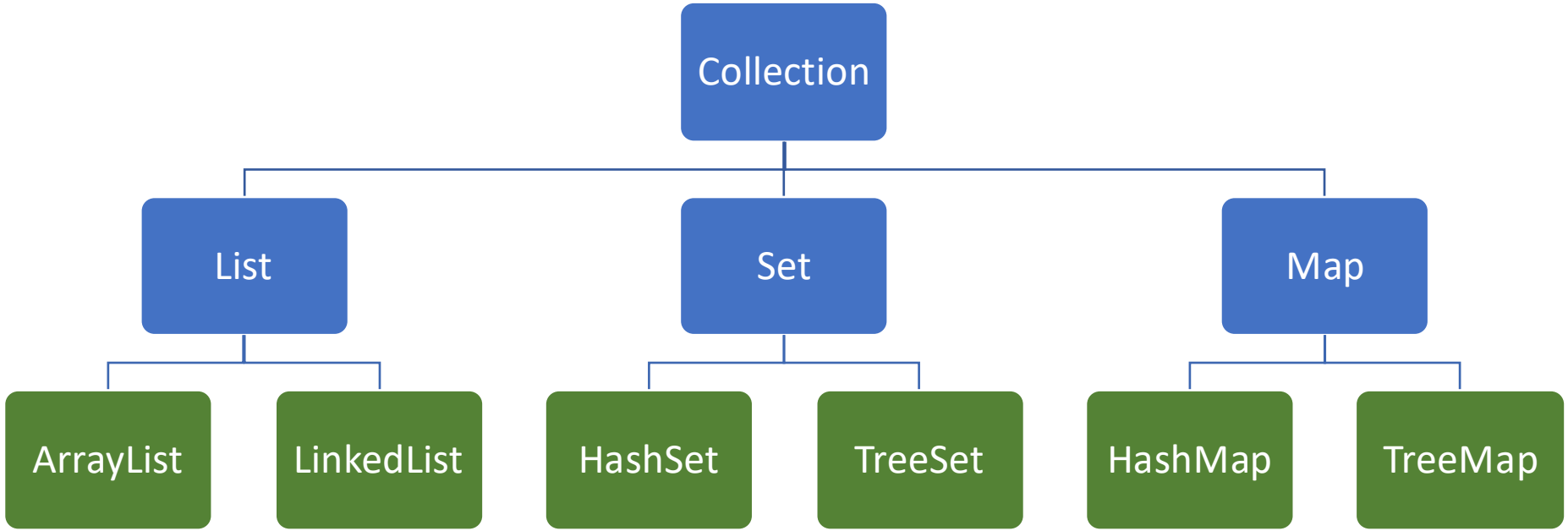
What is a Collection?

```
public interface Collection<E>
extends Iterable<E>
```

The root interface in the *collection hierarchy*. A collection represents a group of objects, known as its *elements*. Some collections allow duplicate elements and others do not. Some are ordered and others unordered. The JDK does not provide any *direct* implementations of this interface: it provides implementations of more specific subinterfaces like `Set` and `List`. This interface is typically used to pass collections around and manipulate them where maximum generality is desired.

- Specifies an API: methods and how they work together, e.g., `.add` and `.addAll`

java.util Classes implementing Collection



Initializing array at scale

- Create and initialize 100 elements

```
int[] a = new int[100];
for(int k=0; k < a.length; k += 1) {
    a[k] = 99;
}
```

- Use java.util.Arrays class

```
Arrays.fill(a,99);
```

Initializing an Object array at scale

- Create and initialize 100 elements

```
String[] sa = new String[100];
for(int k=0; k < sa.length; k += 1){
    sa[k] = new String("hello");
}
```

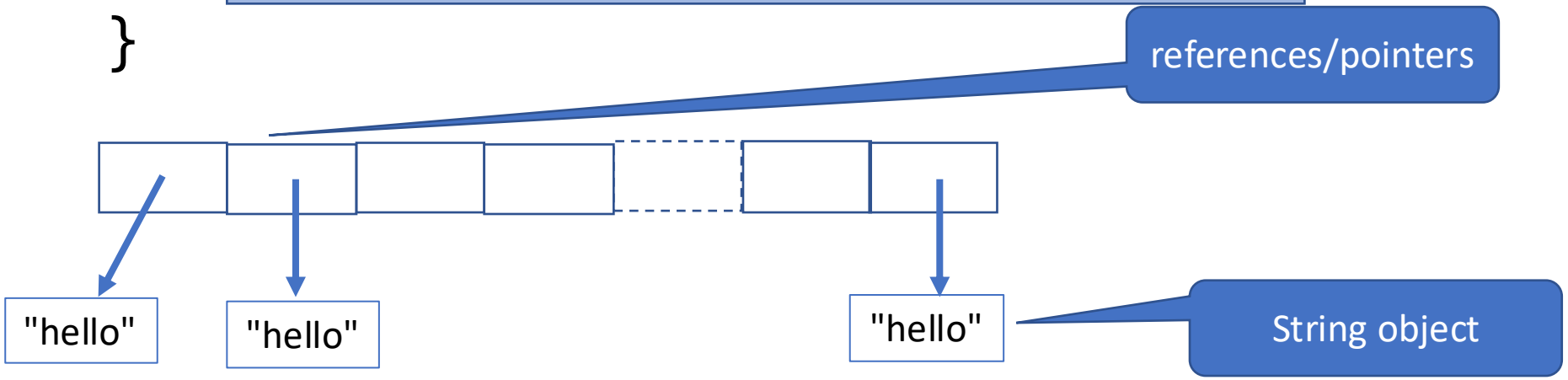
- Use java.util.Arrays class

```
Arrays.fill(sa, "hello");
```

Creating many objects

- 100 String objects created

```
String[] sa = new String[100];
for(int k=0; k < sa.length; k += 1){
    sa[k] = new String("hello");
}
```

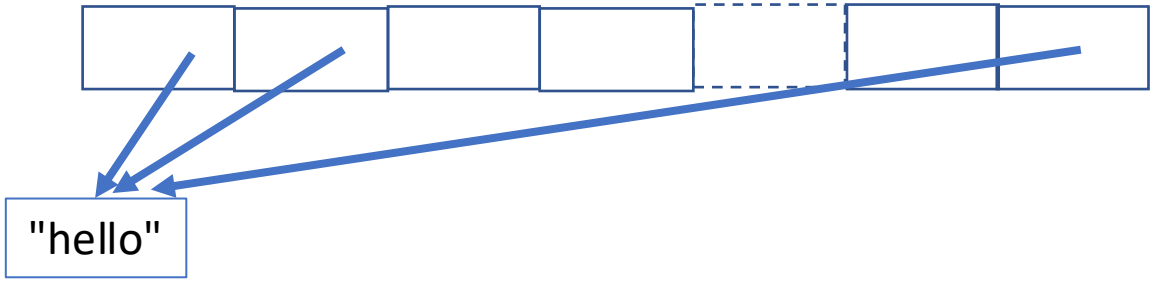


Creating many pointers, one object

- Only one String object created

```
String[] sa = new String[100];
Arrays.fill(sa, new String("hello"))
```

how many times is new called?



Types in an ArrayList

- Like a Java array, homogenous
  - ***Can*** store String, Object, Person201
  - ***Cannot*** store int, double, boolean

- Java has "wrapper" classes
  - `Integer`, `Double`, `Boolean`, `Character`
  - wrap `int`, `double`, `boolean`, `char`

Autoboxing for primitives

- `Integer` is a class, `int` is a primitive
  - int value "autoboxed" into Integer
  - Unboxed as needed
    - `int x = list.get(0);`

```
1    import java.util.ArrayList;
2    int[] a = {10,20,30,40,50};
3    ArrayList<Integer> list =
4              new ArrayList<>();
5    for(int i : a) list.add(i);
```

Classes works as expected with String

- Use java.util APIs with classes
  - Construct ArrayList from …
    - Construct from return value of `asList`
  - Printing array and ArrayList, …

```
1  import java.util.*;
2  String[] a = {"ant", "bat
3  ArrayList<String> list =
4          new ArrayList<>(A
5  System.out.println(a);
6  System.out.println(list);
```

```
System.out.println(a)
[Ljava.lang.String;@1794d431
System.out.println(list)
[ant, bat, yak, zoo]
```

Primitives/Autoboxing not so good

```
ERROR: incompatible types: cannot infer type arguments for java.util.ArrayList<>
    reason: inference variable E has incompatible bounds
      equality constraints: java.lang.Integer
      lower bounds: T,int[]
Rejected field ArrayList<Integer> list
```

- Reminder: can loop over a and add

```
1  import java.util.*;
2  int[] a = {10,20,30,40};
3  ArrayList<Integer> list =
4          new ArrayList<>(Arrays.asList(a));
5  System.out.println(a);
6  System.out.println(list);
```

WOTO 1

APTs are due Thursday (when ??)

- If you have verified algorithm; paper/concept
  - The key word: verified
- And if you've spent >= hour trying to translate algorithm into code

APT Data as of 8:30 pm on Jan 20

	subs <= 2	2 < # <= 4	4 < # <= 6	7 <= # subs
Totality	52	17	15	18
Gravity	65	21	12	7
AccessLevel	64	18	7	14
Starter	47	28	7	11
CirclesCountry	40	16	7	3

Data Structures and Algorithms

- Using ArrayList is reasonably straightforward
- Understanding performance more complex
- 201: analytical and empirical tools for tradeoffs
  - How do ArrayLists "grow" efficiently?

- Levels of abstraction: from high- to low-level

Implementing and Measuring ArrayList

- Performance of an array, but growable
  - Searchable and more, e.g., `list.contains(x)`
  - API provides useful methods for client code

- How to implement? Encapsulate an array!
  - Internally it's an array, but when it's full …. grow!

- Measuring tradeoffs in internal growth

Java ArrayList Source

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/util/ArrayList.java

- private, helper method for public `.add`
  - grow ***doubles size*** of `Object[] elementData`
  - actual code difficult to read/understand

```
482 ⌄      private void add(E e, Object[] elementData, int s) {
483            if (s == elementData.length)
484                elementData = grow();
485            elementData[s] = e;
486            size = s + 1;
487        }
```

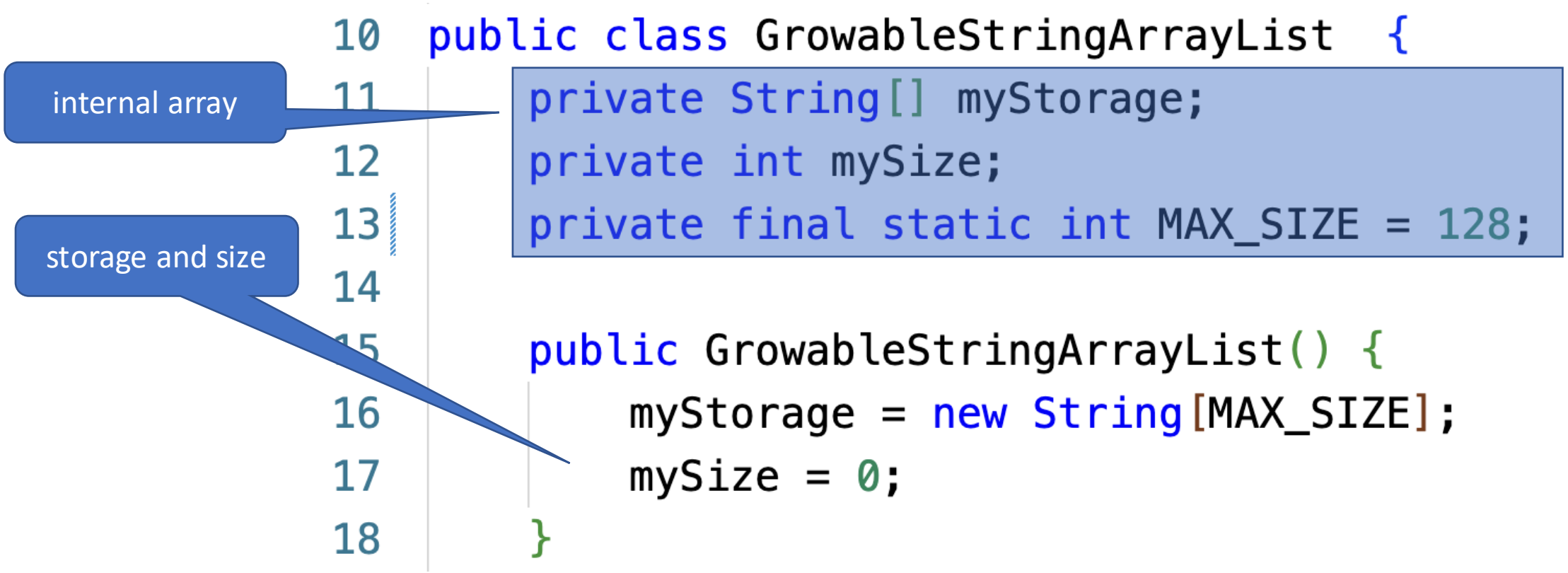
java.util source code is …

- "industrial strength" code takes many special cases into consideration
  - Works in all cases, edge cases, unlikely cases

- To study the concepts we use the diyad library
  - Do It Yourself Algorithms and Data structures

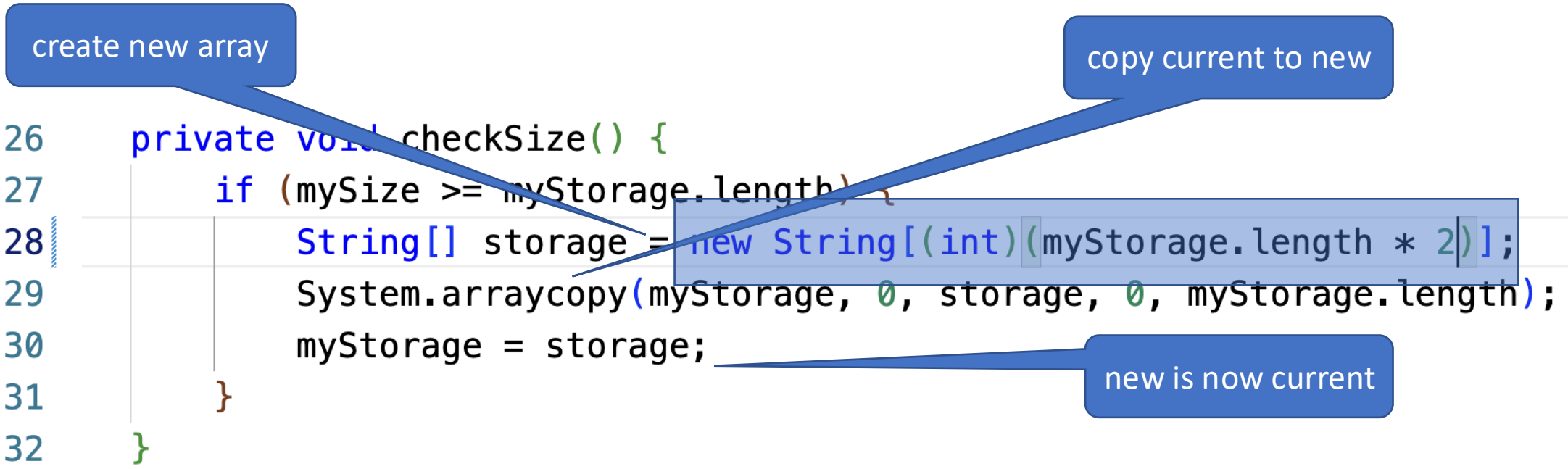
GrowableStringArray class: diyad package

https://coursework.cs.duke.edu/201spring26/diyad/-/blob/main/src/diyad/array/GrowableStringArrayList.java



Growing internal array when needed

- When internal array full? Double size and copy
  - From 128 to 256 to 512 …



Will you always use an API?

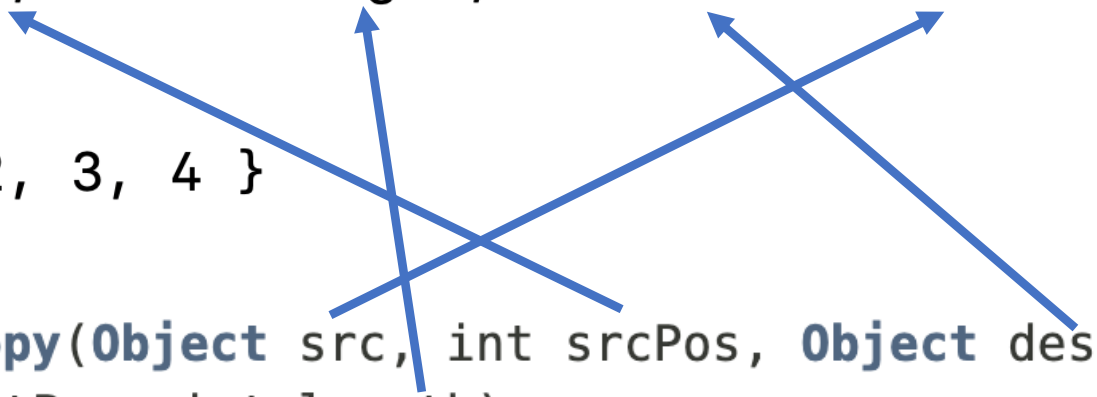
- Loop is in `system.arraycopy`, use the API
  - Here's the explicit loop to copy into new storage

```
35  if (mySize >= myStorage.length) {
36      String[] storage = new String[(int)(myStorage.length * 2)];
37      for(int k=0; k < myStorage.length; k++){
38          storage[k] = myStorage[k];
39      }
40      //System.arraycopy(myStorage, 0, storage, 0, myStorage.length);
41      myStorage = storage;
42  }
```

copy loop explicit

Roll-your-own array copy

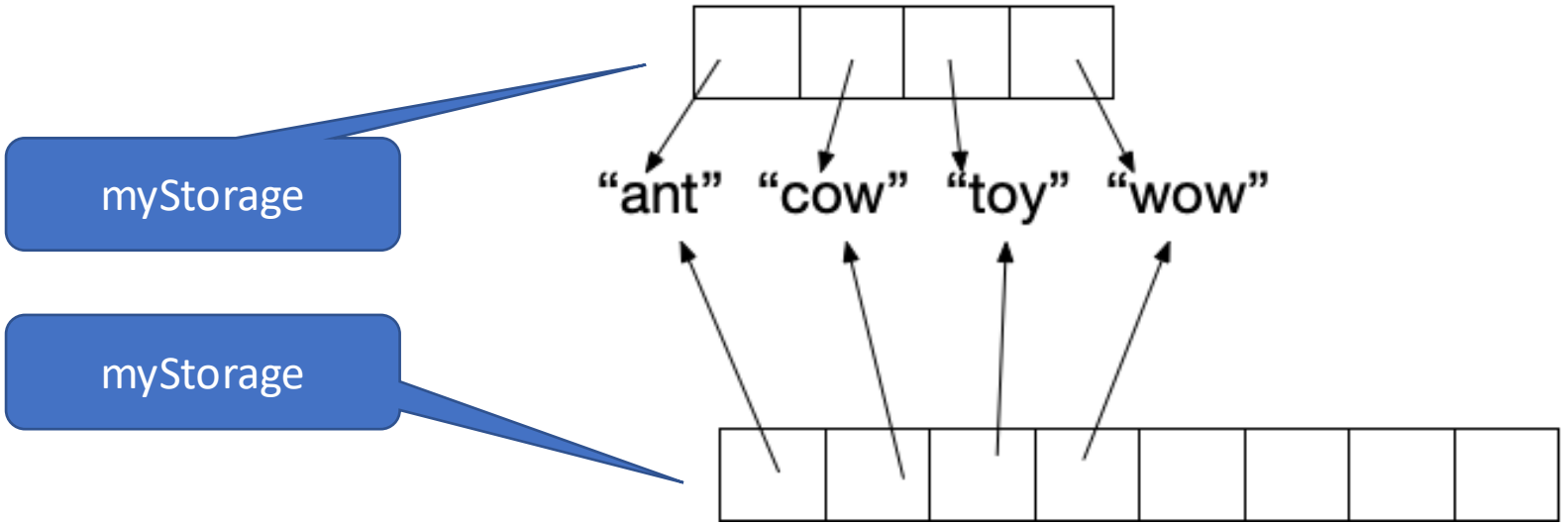
```
jshell> int[] a = {1,2,3,4}
a ==> int[4] { 1, 2, 3, 4 }

jshell> int[] b = new int[4]
b ==> int[4] { 0, 0, 0, 0 }

jshell> for(int k=0; k < a.length; k++) b[k] = a[k]

jshell> b
b ==> int[4] { 1, 2, 3, 4 }
```



```
static void arraycopy(Object src, int srcPos, Object dest,
int destPos, int length)
```

Copying references

- Suppose internal array grows from 4 to 8
  - Create new array with 8 elements
  - Copy four to eight (these are pointers/references)



Benchmarking add elements

- Most `.add` calls are constant time
- But, when array grows? work to do
  - Suppose we grow factor of 1.2 instead of 2?
  - Suppose we grow by adding 1,000 each time?

- Measure empirically and analytically

Benchmarking add elements

- Grow by factor of 2 (left) and 1.2 (right)

size	time
1000000	0.033
2000000	0.039
3000000	0.052
4000000	0.067
5000000	0.117
6000000	0.130
7000000	0.164
8000000	0.235
9000000	0.272
10000000	0.328

size	time
1000000	0.047
2000000	0.104
3000000	0.103
4000000	0.198
5000000	0.271
6000000	0.297
7000000	0.441
8000000	0.412
9000000	0.341
10000000	0.390

Benchmarking again

- Grow by +100,000 and add 10,000

size	time
1000000	0.058
2000000	0.144
3000000	0.350
4000000	0.784
5000000	0.867
6000000	0.991
7000000	1.282
8000000	1.630
9000000	2.028
10000000	2.439

size	time
1000000	0.404
2000000	2.038
3000000	2.731
4000000	3.965
5000000	6.684
6000000	18.819
7000000	18.295
8000000	19.490
9000000	18.730
10000000	28.299

Analytical: Exponential Growth

- Grow by factor of 2 until reaching N
  - 2,4,8,16, 32, …, N
  - 127 = 1+2+4+8+…+64
- Note: sum powers of 2
  - Roughly double last value

$$1 + 2 + 4 + \cdots + N$$

$$= \sum_{i=0}^{\approx \log_2 N} 2^i$$

$$\approx 2N$$

Geometric series formula:

$$\sum_{i=0}^{n} a\, r^i = a\left(\frac{1 - r^{n+1}}{1 - r}\right)$$

Analyze Analytically

- Grow by adding 1,000 until reaching N
  - 1, 1001, 2001, 3001, …, N
  - Arithmetic Series

$$1 + 101 + 201 + \cdots + N$$

Arithmetic series formula:

$$\sum_{i=1}^{n} a_i = \left(\frac{n}{2}\right)(a_1 + a_n)$$

$$= \sum_{i=0}^{\approx N/100} 1 + 100i$$

$$\approx \frac{N^2}{200}$$

What about the details?

- For geometric growth is it 2+4+…+256+…
  - It might be 500, 1000, 2000, 4000
    - 500(1+2+4+…)

- For arithmetic growth is it 1+1001+2001+…
  - It depends on how much we add each time

- In the limit ***coefficients don't matter***
  - This is ***linear***: N or ***quadratic*** N^2

Math in 201

- You will not need to prove formally
  - See 230 and 330 in compsci

- Do expect you to recognize and use formulae
  - Geometric growth is ***linear***
  - Arithmetic growth is ***quadratic***

WOTO 2

Dissecting Person201: related classes

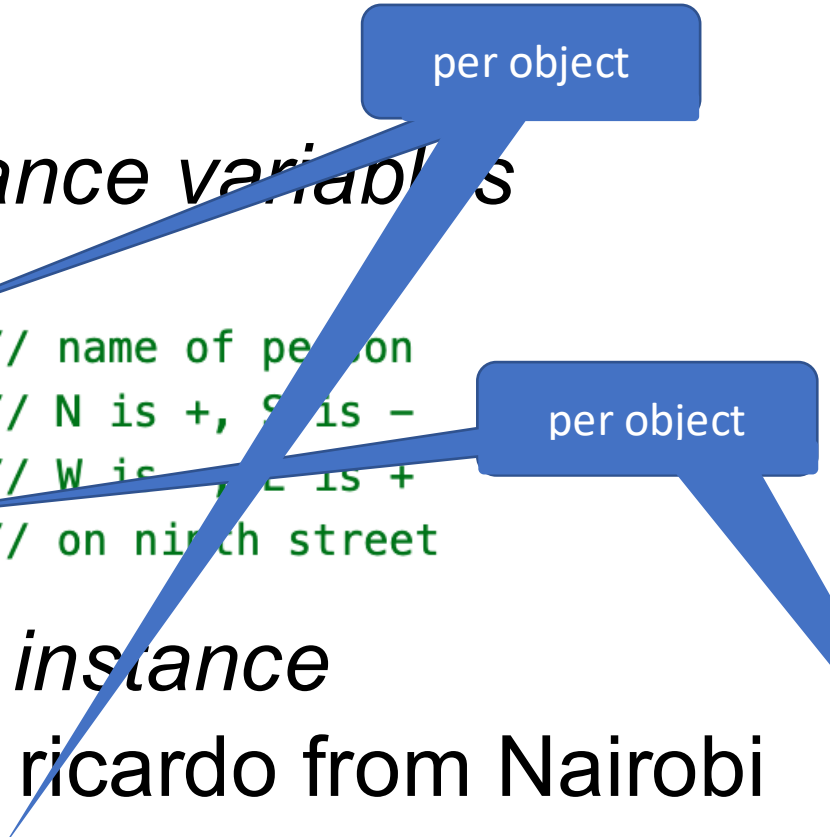
- Exploring Java and programming in context
  - Use P0-related classes to understand Java

- Be curious about the code you write
- Be curious about the code you call
- Use LLMs (or the Internet)

Person201: state and behavior

- State specified by *instance variables*

```
11 public class Person201 {
12     private String name;      // name of person
13     private double latitude;  // N is +, S is -
14     private double longitude; // W is -, E is +
15     private String eatery;    // on ninth street
```



- Specific to each object *instance*
  - claire from Oakland, ricardo from Nairobi

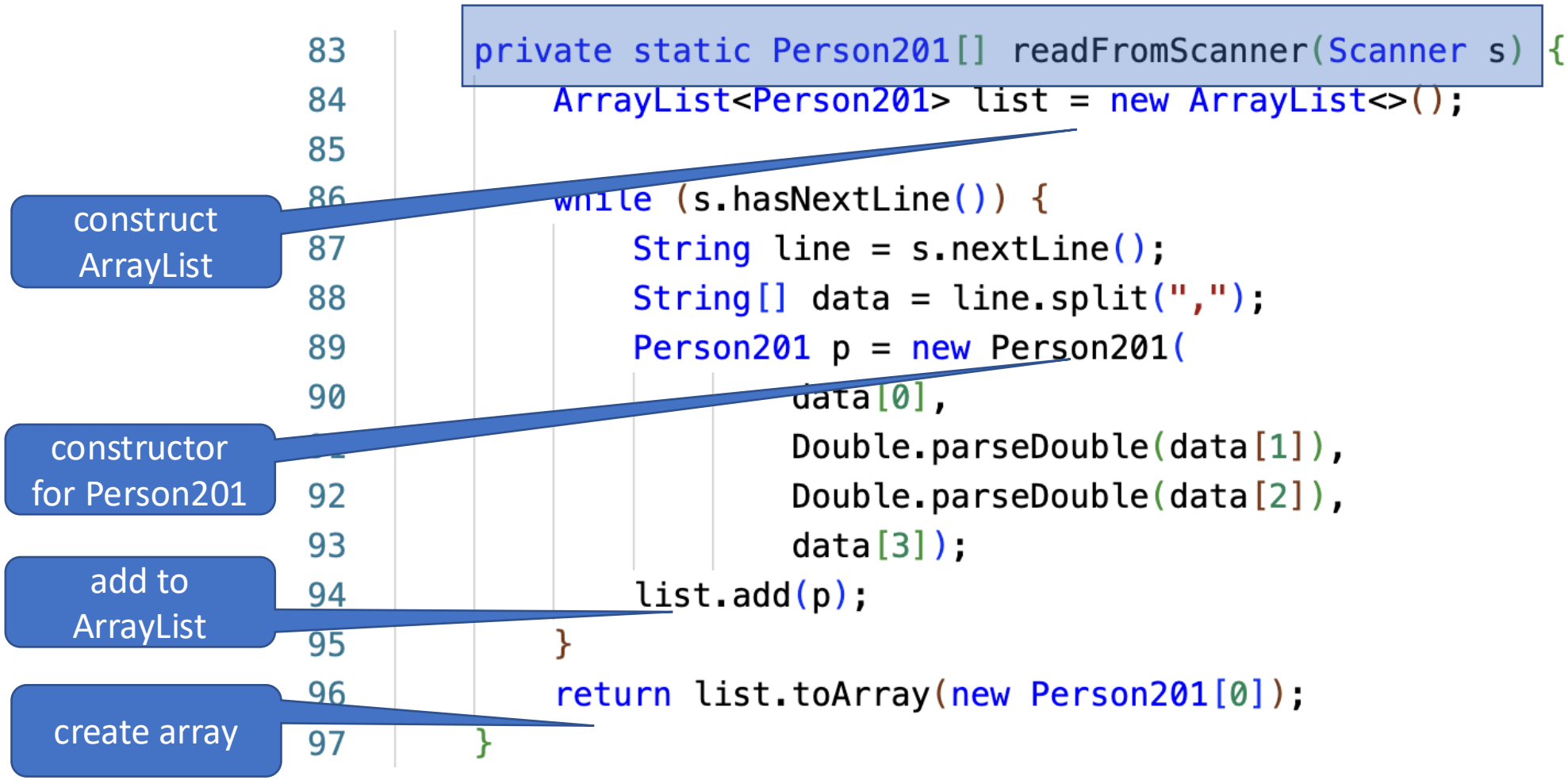
```
Person201 a = new Person201("claire",37.8044,-122.2712,"Vin Rouge");
Person201 b = new Person201("ricardo",-1.2921, 36.8219,"Elmo's Diner");
```

Reading lots of data from files/URLs

- Processing an array of Person201 objects
  - Read using Person201Utilities methods

```
63      public static Person201[] readFile(String fname) throws IOException {
64          Scanner s = new Scanner(new File(fname));
65          Person201[] result = readFromScanner(s);
66          s.close();
67          return result;
68      }
```

Person201Utilities code




```
83      private static Person201[] readFromScanner(Scanner s) {
84          ArrayList<Person201> list = new ArrayList<>();
85
86          while (s.hasNextLine()) {
87              String line = s.nextLine();
88              String[] data = line.split(",");
89              Person201 p = new Person201(
90                      data[0],
                        Double.parseDouble(data[1]),
92                      Double.parseDouble(data[2]),
93                      data[3]);
94              list.add(p);
95          }
96          return list.toArray(new Person201[0]);
97      }
```

`private static Person201[]`

- In contrast with public
  - This is a helper method, callable within class only

- Belongs to the class, not to an object
  - Math.sqrt(25); No ***per object***

```
jshell> new Math()
|  Error:
|  Math() has private access in java.lang.Math
|  new Math()
|  ^--------^
```

PeopleDownloader.java

```
Run | Debug
77  public static void main(String[] args){
78      try {
79          Person201[] pa = PeopleDownloader.loadData(URL);
80          for(Person201 p : pa) {
81              System.out.println(p);
82              //System.out.printf("%s,%2.3f,%2.3f,%s\n",p.name(),p.la
83          }
84          System.out.printf("total = %d\n",pa.length);
85      }
86      catch (Exception e) {
87          System.err.println("problem loading data");
88          e.printStackTrace();
89      }
90  }
```

static method

.toString()

alternative?
method throws

Where do you/will you eat on 9th Street?

- Live Git and Think, use zip file as alternative
  - Autograder and Analysis via Gradescope

ACM Grace Murray Hopper Award

- Outstanding young (<= 35) computer professional of the year, on the basis of a single major

2017

Amanda Randles

Duke UNIVERSITY

Craig Gentry

2011

Luis von Ahn

Luis von Ahn

- Zooming in @ 3:05 on Thursday, January 22
  - Compsci 342, Griffith Theatre

- Story of reCaptcha, DuoLingo, the role of AI
  - Starting before Compsci 201, but including 201