ADTs, Hashing, Gitlet Intro

Quote of the week: “More than cleverness, we need kindness and gentleness.”

ADTs

- Abstract Data Types, or **ADTs**, are purely theoretical descriptions of data structures
- Described *only* in terms of behavior and the operations they support

We’ve learned a lot about lists…

- We learned two *implementations* of the list, one with linked nodes and one with arrays
- We know that lists exist in different programming languages (Python, Java, …)
- The *idea* of the list remains constant throughout

The list (sequence)

- The List ADT supports (roughly) the the following operations:
  - **Create** an empty list
  - **Add** an item to the front or back, or at a numbered location
  - **Get** an item from a numbered location

The list (sequence)

- You can define more complex operations on lists by combining the operations from before
  - e.g. You can check if a list contains an item by getting the item at position 0, 1, 2, 3… up to size

Rapid-fire ADTs

- List
- Set
- Stack, Queue, Priority Queue
- Map
- Tree
- Graph

The set

- Sets typically have the following operations
  - **Create** an empty set
  - **Add** an item to the set
  - **Check** if the set already contains an item

The set

- Consequences of this behavior:
  - Compared to the list, the set is **unordered**…
  - …and **does not contain duplicates**

The stack

- Supports the following operations:
  - **Create** an empty stack
  - **Add** an item to the stack
  - **Check** the *most recent* item added to the stack
  - **Remove** the *most recent* item added

http://www.clipartpanda.com/categories/stack-of-books-clipart

The queue

- Supports the following operations:
  - **Create** an empty queue
  - **Add** an item to the queue
  - **Check** the *oldest* item added to the queue
  - **Remove** the *oldest* item added

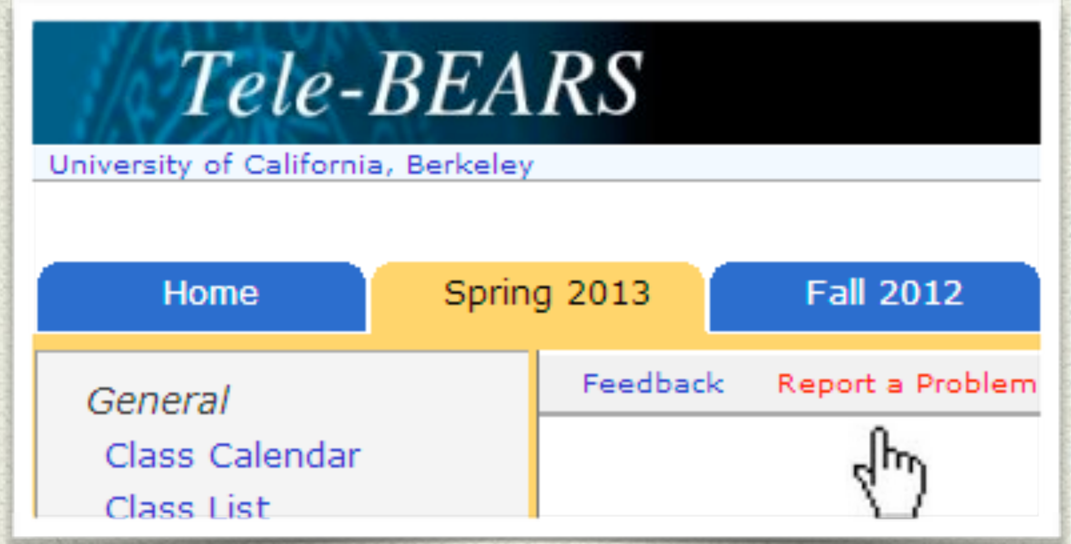
Opposite of a stack!

http://cdn2.benzinga.com/files/imagecache/1024x768xUP/images/story/2012/shutterstock_85396711_0.jpg

The priority queue

- Supports the following operations:
  - **Create** an empty queue
  - **Add** an item to the queue
  - **Check** the *most important* item added to the queue
  - **Remove** the *most important* item added

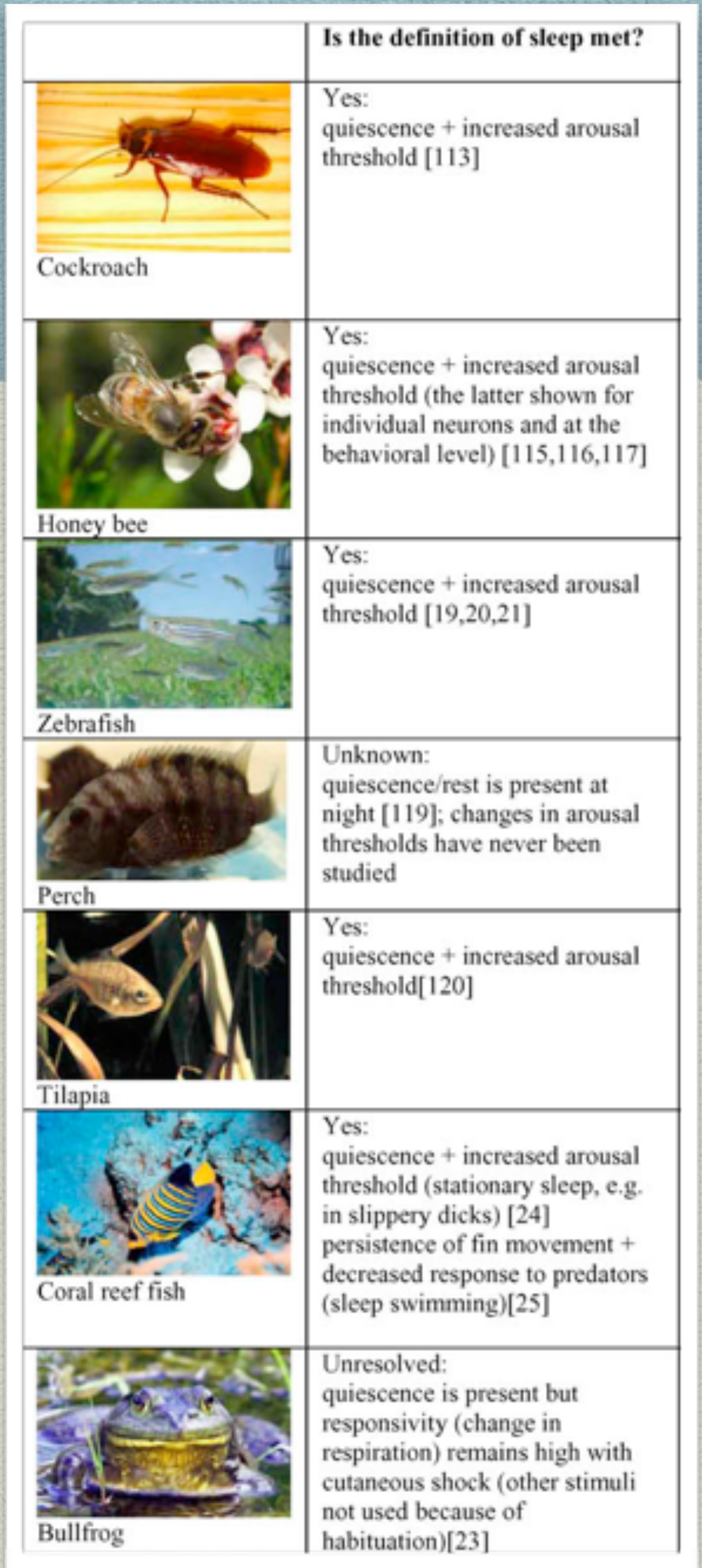
What does this mean? This is one of the more complicated ADTs. We'll save this one for later.



Everyone's favorite priority queue http://clog.dailycal.org/tag/tele-bears-oracle/

The map

- aka a **Dictionary** or **Lookup Table**
- Supports the following operations:
    - **Create** an empty map
    - **Add** a key with a value to the map
    - **Lookup** the value of a given key
    - **Change** the value of a key in the map

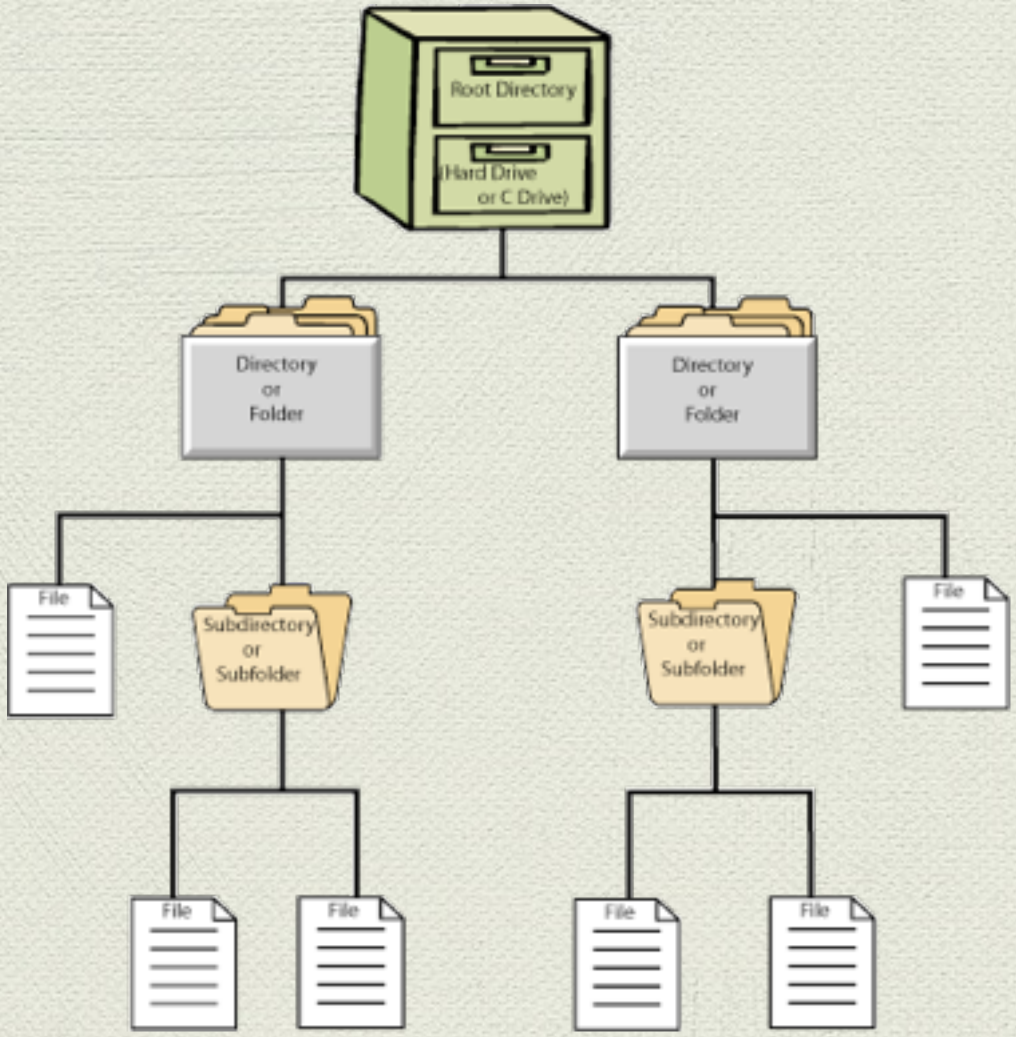


	Is the definition of sleep met?
Cockroach	Yes: quiescence + increased arousal threshold [113]
Honey bee	Yes: quiescence + increased arousal threshold (the latter shown for individual neurons and at the behavioral level) [115,116,117]
Zebrafish	Yes: quiescence + increased arousal threshold [19,20,21]
Perch	Unknown: quiescence/rest is present at night [119]; changes in arousal thresholds have never been studied
Tilapia	Yes: quiescence + increased arousal threshold[120]
Coral reef fish	Yes: quiescence + increased arousal threshold (stationary sleep, e.g. in slippery dicks) [24] persistence of fin movement + decreased response to predators (sleep swimming)[25]
Bullfrog	Unresolved: quiescence is present but responsivity (change in respiration) remains high with cutaneous shock (other stimuli not used because of habituation)[23]

Map of animals to whether they sleep or not. From: https://askabiologist.asu.edu/plosable/who-needs-sleep-anyway

The tree

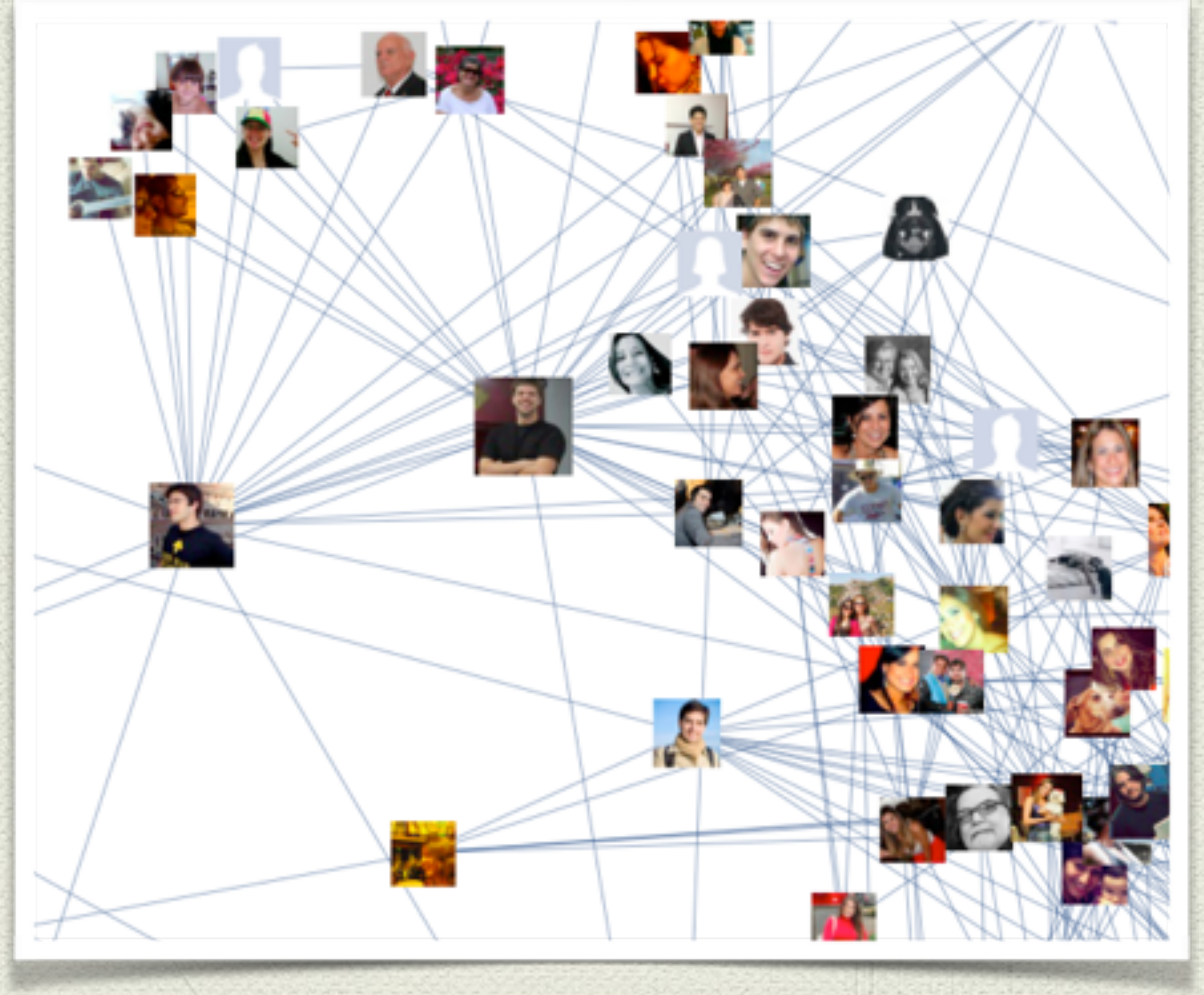
- Organizes data *hierarchically*
- Supports the following operations
  - **Create** a tree with a root node
  - **Add** a child to a node
  - **Get** all the children of a node



https://www.cs.colostate.edu/~cs155/Fall15/Lecture1

The graph

- Supports the following operations
  - **Create** an empty graph
  - **Add** a vertex to the graph
  - **Add** an *edge* (connection) between two vertices
  - **Get** all the neighbors of a vertex



http://mathematica.stackexchange.com/questions/11673/how-to-play-with-facebook-data-inside-mathematica

Utility operations

- There’s flexibility in the kinds of operations each ADT could have
- Most would be typically assumed to be able to
  - Check how many items it has
  - Remove an item
  - Iterate through each item

ADTs — what's the point?

- You can think about how to solve problem purely using ADTs
- The implementation details might not be a big deal

Example problem

- Given a string with parentheses in it, e.g. “ (((( ) ( ))) ( )) ”, describe an algorithm that determines if they are correctly balanced
  - “ (( ) ( )) ” is correctly balanced
  - “ (( ) (“ is not

Example problem

- Solution: Every time we see “ (“, we are descending a level into the expression. Every time we see “) ”, we are ascending back a level
- To be balanced, we must start and end at level 0, and never hit 0 otherwise
- We can use a **stack**. When we find “ (“, add an item on the stack. When we find “) ”, take an item off the stack. If the stack becomes empty at the end of the expression, and not before, we succeed

The point

- I can describe the solution to the previous problem purely theoretically
- Now that you know this, you could easily write it in Java, Python, etc., using a stack in that language

Fun with ADTs

- Quiz time!
- (So early??)

Fun with ADTs

- Imagine you have a class `Stack`, with a constructor the following methods
  - `void push(int item)` — adds an item to the stack
  - `int pop()` — removes and returns the most recently pushed item
  - `boolean isEmpty()` — checks if it has any items

Fun with ADTs

- Your task is to write a class `Queue`, with a constructor the following methods
  - `void enqueue(int item)` — adds an item to the queue
  - `int dequeue()` — removes and returns the oldest enqueued item
- **With the following caveat**: The class can *only* use two types of variables: `int`, and `Stack`.

One solution

- (Not the most efficient solution)

```java
import java.util.Stack;
public class Queue {

    private Stack<Integer> myItems;

    public Queue() {
        myItems = new Stack<>();
    }

    public void enqueue(int i) {
        myItems.push(i);
    }

    public int dequeue() {
        Stack<Integer> tempStack = new Stack<>();
        while (!myItems.isEmpty()) {
            tempStack.push(myItems.pop());
        }
        int results = tempStack.pop();
        while (!tempStack.isEmpty()) {
            myItems.push(tempStack.pop());
        }
        return results;
    }
}
```

Food for thought…

- Can this quiz be done using only one stack?
- Answer might be more interesting than you think…

ADTs in Java

- Are commonly represented as **interfaces**
    - Interface specifies only *behavior*, not *implementation*
    - e.g. `List` is an interface, `ArrayList` and `LinkedList` are implementations!

Let’s make a map!

- A **map** is essentially just a **set** of key-value pairs…

Let’s make a set!

- First idea: Use an array, or ArrayList, or LinkedList

Let's make a set!

- **Create** an empty set: `myValues = new ArrayList<E>();`
- **Add** an item to the set: `myValues.add(E item);`
- **Check** if the set already contains an item: iterate through the `myValues` until we find the item

Runtimes?

- **Create** an empty set: `new ArrayList<E>();` `O(1)`

- **Add** an item to the set: `add(E item);` `O(1)` usually

- **Check** if the set already contains an item: iterate through the list until we find the item `O(location of item)`, or `O(n)` if not in set, if set has `n` items

This is really bad

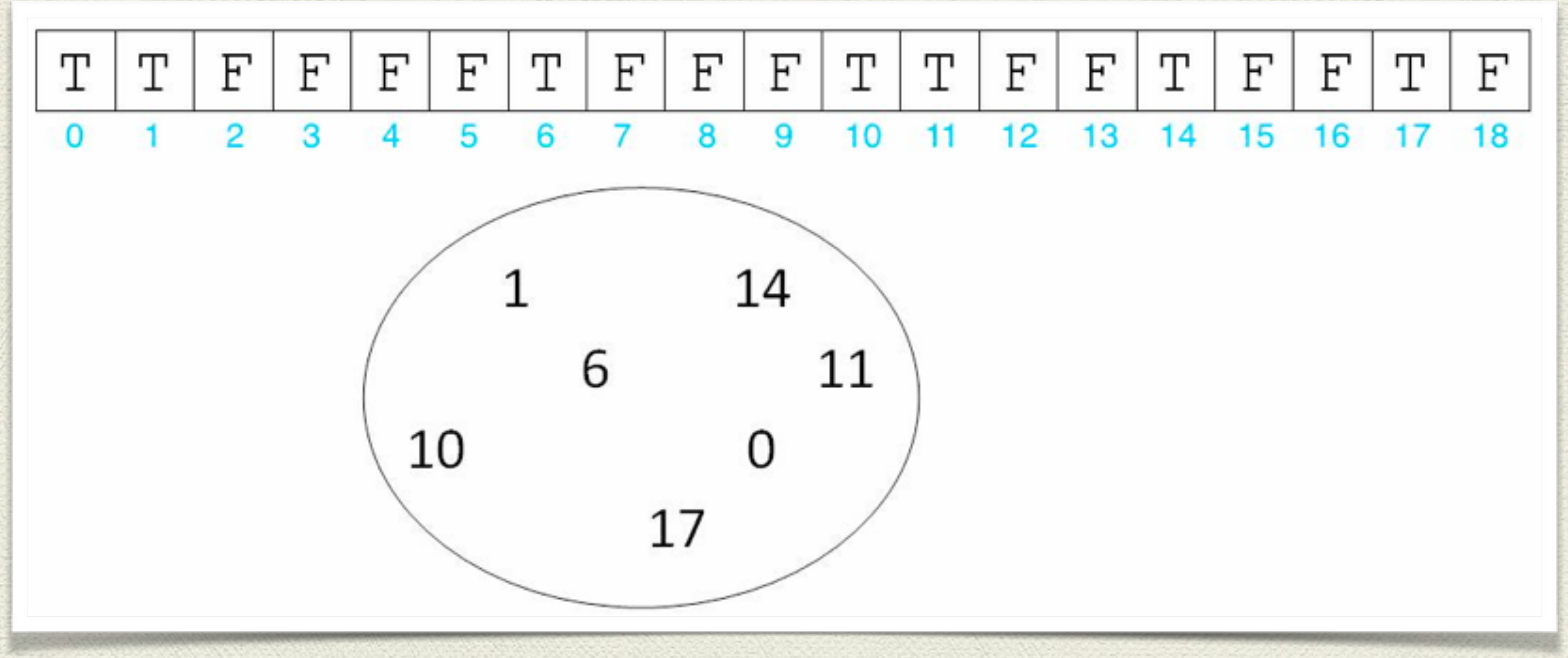
- We have to iterate through the ArrayList just to check if the set contains one item
- And every time we want to check if the set does *not* contain an item, we have to iterate through the whole thing!

Goal:

- O(1) runtime for checking if the set contains an item
- Can it be done…?!
- (drumroll…!)

Goal: O(1) contains

- You already did this in lab.



T	T	F	F	F	F	T	F	F	F	T	T	F	F	T	F	F	T	F
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

- Remember this thing?
- This is not a set of booleans
- It's a set of integers!
- The things the set contains are actually the indices of this array

The set of integers

- **Create** an empty set: make a new array of some big size named `contains`

- **Add** an integer item to the set:
  `contains[item] = true;`

- **Check** if the set already contains an item:
  `return contains[item];`

Runtimes?

- **Create** an empty set: make a new array of some big size Eh, not really important

- **Add** an item to the set: `contains[item] = true;` O(1)

- **Check** if the set already contains an item: `return contains[item];` O(1)!!

But there's a problem

- How do we make a set of something other than integers?
- Ex: How do we make a set of strings?

This doesn’t really make sense

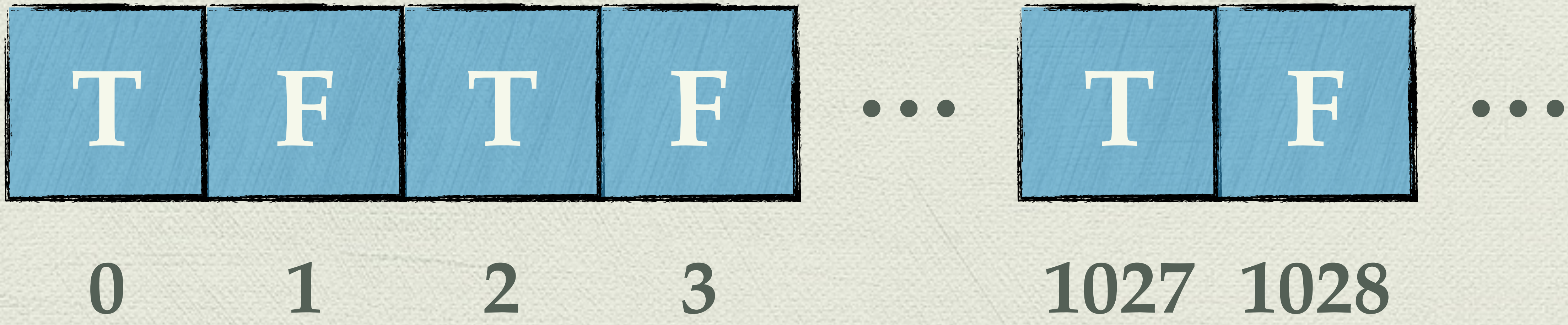
T	F	F	T
“kindness”	“cleverness”	“machinery”	“gentleness”

- Is this a set containing “kindness”, “gentleness”, but not “cleverness” or “machinery”?
- Can’t index into an array at a String…

Idea:

- Associate each String with a number
- “`a`” will be 0, “`b`” will be 1, “`c`” will be 2, “`gentleness`” will be something really, really big…

Well, at least it works?



T	F	T	F	…	T	F	…
0	1	2	3		1027	1028	

- This a set containing “`a`”, “`c`”, and “`gentleness`”, but not “`b`” or anything else
- Assume the number of “`gentleness`” is 1027

Well, at least it works?

- **Create** an empty set: make a new array of some REALLY big size
- **Add** a String item to the set:
  `contains[item.getNumber()] = true;`
- **Check** if the set already contains an item:
  `return contains[item.getNumber()];`

Runtimes?

- **Create** an empty set: make a new array of some REALLY big size Uh, is this a problem now?

- **Add** a String item to the set: `contains[item.getNumber()] = true;` O(1), assuming we can figure out the number of a String in constant time

- **Check** if the set already contains an item: `return contains[item.getNumber()];` O(1), assuming as above

Actual problem

- **Create** an empty set: make a new array of some REALLY big size
- We can't possibly make an array big enough to hold every possible String!
- We have to store a `false` value for *every single possible String* that's not in our set!

New idea:

- Create an array of some fixed, medium size
- **Mod** the number of our String by the size of the array, and store at that location
    - *Think about it:* (the result of `x % n` is guaranteed to be `< n`)

Mod in action

T	F	F	T
0	1	2	3

- Say the number of “gentleness” is 1027.
  - 1027 % 4 = 3, so we check position 3
- Pretty cool, right? We can still predict the index the String would appear at, allowing us to check in constant time.

It kinda works?

- **Create** an empty set: make a new array `contains` of some moderate size

- **Add** a String item to the set:

  ```
  contains[item.getNumber() % contains.length] = true;
  ```

- **Check** if the set already contains an item:

  ```
  return contains[item.getNumber() % contains.length];
  ```

Problem

- (More problems?!)

Problem

- Even if all Strings have a unique number, those numbers modded could end up the same…
- This is called a **collision**

Problem: Collisions

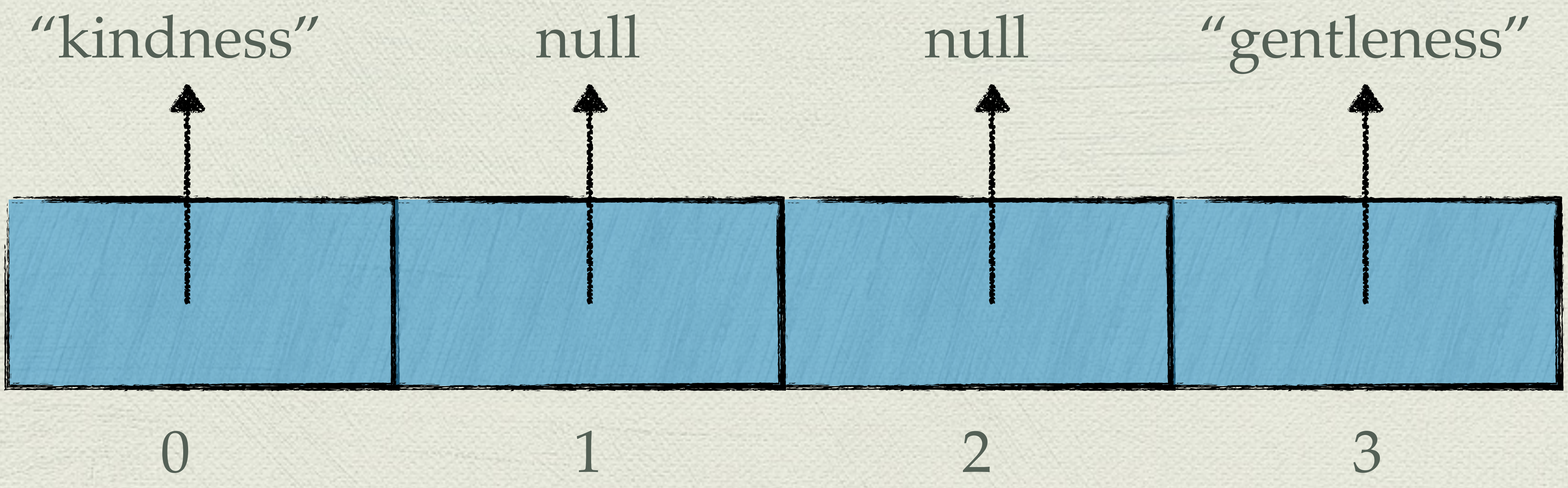
T	F	F	T
0	1	2	3

- This is a set that contains “kindness” and “gentleness”
- Does it contain “d”?
- If the number of “d” is 3, then 3 % 4 = 3, so it looks like it does!
- But it's not supposed to…

Idea:

- Store the actual String instead of just `true`. Consider null be to `false`.

Problem: Collisions



- This is a set that contains "kindness" and "gentleness"
- Does it contain "d"?
- Now we can tell it doesn't!

It kinda works?

- **Create** an empty set: make a new array `contains` of some moderate size

- **Add** a String item to the set:

```
contains[item.getNumber() %
contains.length] = item;
```

- **Check** if the set already contains an item:

```
return contains[item.getNumber() %
contains.length].equals(item);
```

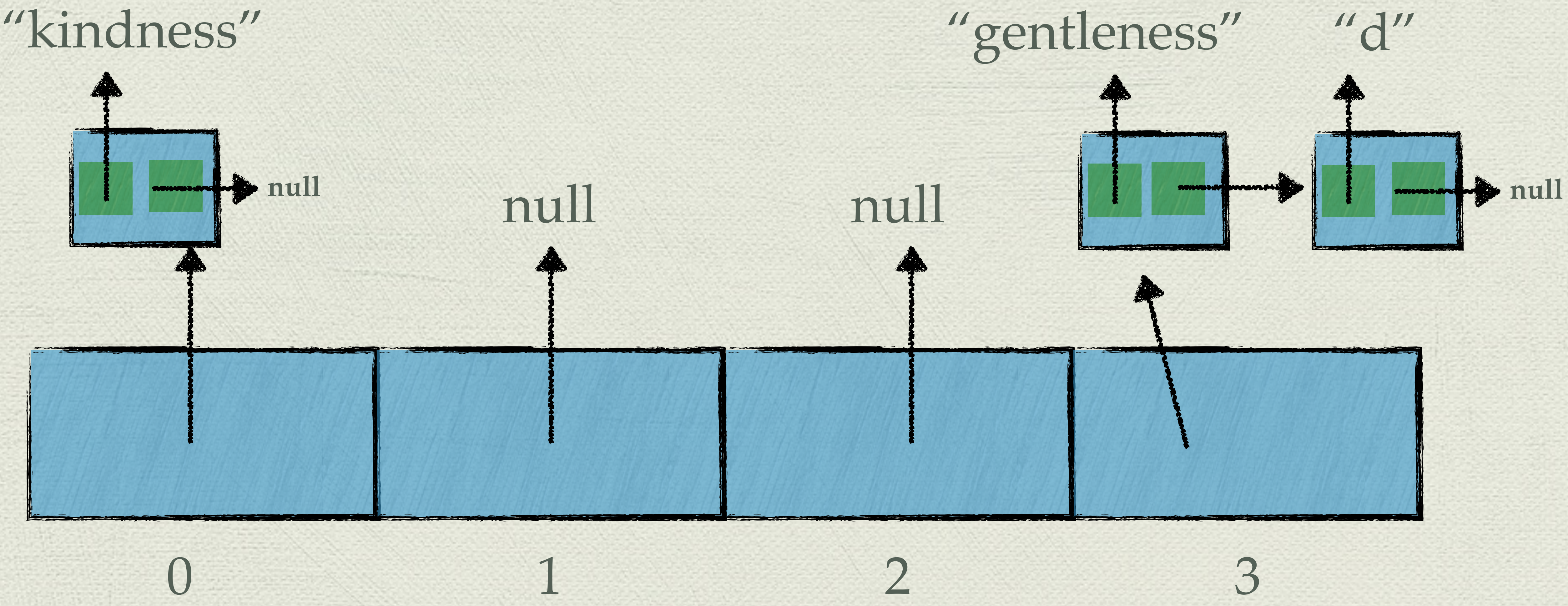
Problem

- Well, what if we wanted to store *both* “gentleness” and “d”?

Idea

- Store multiple things by… storing a list

Problem: Collisions



- This is a set that contains "kindness", "gentleness", and "d"

It works!!!

- **Create** an empty set: make a new array `contains` of some moderate big size

- **Add** a String item to the set:

  ```
  contains[item.getNumber() %
  contains.length].add(item);
  ```

This is the contains method of the LinkedList class. Not the name of our array.

- **Check** if the set already contains an item:

  ```
  return contains[item.getNumber() %
  contains.length].contains(item);
  ```

Problem

- **Check** if the set already contains an item:

```
return contains[item.getNumber() %
contains.length].contains(item);
```

Runtime of this is… proportional to the number of things in the list?!

We lost O(1)…

But I’m all out of ideas

- Runtime is **NOT** guaranteed O(1)
- Instead, **it depends on the number of collisions**
- Luckily, the number of collisions should much smaller than the total number of items in the set

Collisions

- Can reduce number of collisions by making the array bigger
- But we might not have the amount of memory for it.
- **Idea**: Expand the array if the number of collisions gets too high

The hash table

- This thing we just invented is called a **hash table**

- Key feature: supports *nearly* O(1) contains checking by associating each thing in it with a special number called a **hash code**

- The `.getNumber()` method of String is actually called a **hash function.** In Java, the real method is `.hashCode()`

Story of a beautiful partnership

- Last lecture, we witnessed a great battle between arrays and linked lists, to decide which was better for a sequence

- A hash table turns out to be *an array of linked lists*.

- The array gives us fast indexing, and the linked lists gives us guaranteed fast appends

- It turns out the best thing was when the array and linked list worked together!

The hash table in Java

An interface (ADT)

```
Set set = new HashSet();

set.add(“noodles”);

set.add(“macaroons”);

System.out.println(set.contains(“no
odles”)); // true!
```

The actual class (implementation)

Let’s talk about .hashCode()

- I told you String has `.hashCode()` method, which returns a unique number for the String
- It’s not truly unique, but pretty close
- How do we write this?

Simple String .hashCode()

- Goal: Associate a number with a String

```
public int hashCode() {
  int hash = 0;
  for (int i = 0; i < s.length(); i++) {
    hash += s.charAt(i);
  }
  return hash;
}
```

- Luckily, each `char` already has a number associated with it. We could use that.

Actual String .hashCode()

- Or, closer to it anyway

```
public int hashCode() {
  int hash = 0;
  for (int i = 0; i < s.length(); i++) {
    hash = 31 * hash + s.charAt(i);
  }
  return hash;
}
```

- Why so complicated? Turns out it reduces collisions. Don’t worry about this. More of a CS 70 topic

Let’s talk about .hashCode()

- So our hash table stores Strings
- What if we want to store some other kind of Object?
- All it needs is a `.hashCode()` method.

Let's talk about .hashCode()

- The `Object` class has a `.hashCode()` method. So all objects do!

- `Object`'s `.hashCode()` is useless. Expected to be overridden.

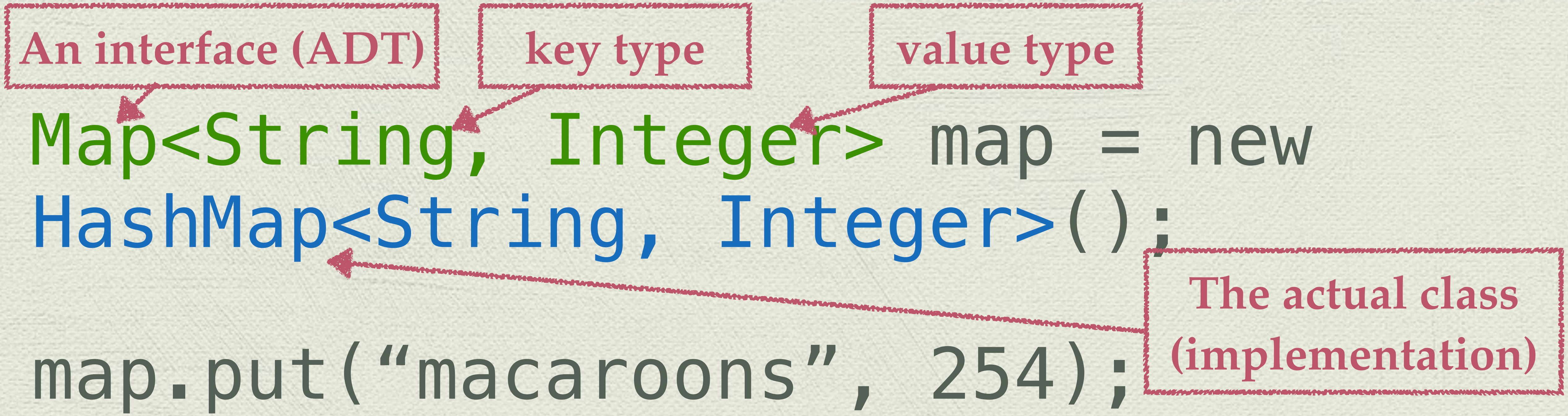
    - Just like `.equals` and `.toString()`

    - Every class you write should have its own `.hashCode()` method, its own way of turning itself into a number

Hash Maps

- I just described how to use a hash table to make a set.
- You can also use it to make a map.
- Instead of storing items, just store key-value pairs together

The hash map in Java



```
Map<String, Integer> map = new
HashMap<String, Integer>();

map.put(“macaroons”, 254);

map.put(“noodles”, 2);

System.out.println(map.get(“macaroo
ns”)); // 254
```

Hash tables

- Are used everywhere all the time
- Probably the most useful non-obvious data structure in this course
- Java's real hash table is a little fancier than I've shown you, but I got you to the basic idea

BREAK

Intro to project 2

- In project 2, you'll be building a simpler version of the popular *version control software,* **git**
- It's called gitlet

Version control software?

- Version control software helps you maintain different backups of files on your computer
- Specifically, you could maintain different backups of code you write
- That way, you can revert to old versions if you like

Files, huh?

- In this project, you'll be working with *your computer's file system*, which is something you might not have done before
- So I provide a brief intro

Files, huh?

- Whenever you run a Java program, you run it from a **certain location** in your file system
- You can access files only in that folder by their name…

File System Demo

Files, huh?

- To access files in a folder, you have to include the folder name in front
  - The complete list of folders and file name is referred to as the **path** to the file

Introducing the File class

- The `File` class in Java allows us to easily manipulate files
- `File f = new File(“values/deep/kindness.txt”);`
  - Does **<u>NOT</u>** create a new file on our computer. It only gives us a variable that allows us to manipulate the existing file

Introducing the File class

- The `File` class has a bunch of useful methods! Explore them on your own.

Backups, huh?

- If we’re going to maintain information about backups on our computer, we need to **save the state** of our program
- Normally, when you run a Java program, all objects are garbage collected at the end and disappear

Backups, huh?

- The way to save state on a computer is using a file
- We want some way to **save our Java objects to a file**!

Persistent List Demo

Persistent List

- How did I do this?
- Using the **`Serializable` Interface**
- Any object that implements `Serializable` can be saved to a file, then loaded back in the next time we run Java

Serializable

- So, what methods are required to implement Serializable?
- *None.*
- What.

Serializable

- As long as a class and all of its instance variables implement Serializable, you can save it to a file

- How does this work? Magic. Don't worry about it.