LEC 15

CSE 122

Collections

Questions during Class?

Raise hand or send here

sli.do #cse122

BEFORE WE START

Talk to your neighbors:

What plans do you have for Spring break? Going anywhere fun?

Music: 122 24wi Lecture Tunes

Instructors Miya Natsuhara and Joe Spaniac

TAs

Ailsa	Chaafen	Helena	Megana	Sahej
Alexander	Chloe	Jessie	Mia	Shivani
Ambika	Claire	Katharine	Minh	Smriti
Andy	Colin	Kavya	Nicolas	Steven
Arkita	Colton	Ken	Poojitha	Vinay
Atharva	Connor	Kyle	Rohini	Zane
Autumn	Elizabeth	Logan	Ronald	
Ayush	Hannah	Marcus	Rucha	

Lecture Outline

- Announcements
- Optional
- Recap of Collections
- Dumb Data Structures
- Collections

Announcements

- Resubmission Cycle 5 (R5) out; due February 27th by 11:59 PM
- Programming Assignment 3 (P3) out tonight!
  - Due February 29th by 11:59 PM
- Quiz 2 Thursday, February 29th
  - Same day as P3, similar to Quiz 0 – plan accordingly!
- Reminder on Final Exam: **Wednesday, March 13th 12:30 – 2:20 PM**

Lecture Outline

- Announcements
- Optional
- Recap of Collections
- Dumb Data Structures
- Collections

First, a quick correction:

Square is a company

Optional

`Optional` is a Java class that is used to handle situations where a value is *sometimes* there.

Like a collection, Optional uses <> to denote the type it contains..

e.g., `Optional<String>, Optional<Integer>, Optional<Point>`

Optional Methods

Method	Description
`Optional.empty()`	Creates an empty `Optional` object
`Optional.of(…)`	Creates an `Optional` object holding the object it's given
`isEmpty()`	Returns `true` if there *is no* value stored, and `false` otherwise
`isPresent()`	Returns `true` if there *is* a value stored, and `false` otherwise
`get()`	Returns the stored object from the `Optional` (if one is stored; otherwise throws a `NoSuchElementException`)

The Optional class has more than just these methods, but these are what you'll need to focus on for this class!

Optional Methods

`isEmpty()`, `isPresent()`, and `get()` are called like normal instance methods (on an actual instance of `Optional`).

`Optional.of(…)` and `Optional.empty()` are called differently

(Like the `Math` class methods)

Why Optional?

Using `Optional` can help programmers avoid `NullPointerExceptions` by making it explicit when a variable may or may not contain a value.

- Remember – `null` refers to the absence of an object!

There are other `Optional` methods (that you should explore in your own time if you're interested) that can be really useful to cleanly work with data that may or may not be present.

Student / Course Example one more time…

Let's add two more methods to `Course.java`:

```
public void setCourseEvalLink(String url)

public Optional<String> getCourseEvalLink()
```

The link to the evaluations for a course doesn't usually exist until the last few weeks of the quarter. What if a client calls `getCourseEvalLink` before one is set up?

`Optional` to the rescue!

Lecture Outline

- Announcements
- Optional
- **Recap of Collections**
- Dumb Data Structures
- Collections

Goal for Today

Review some of the data structures we've talked about this quarter

Understand how Java organizes them with *interfaces*

Collections: What *classes* have we seen so far?

…

Collections: What *interfaces* have we seen so far?

…

Lecture Outline

- Announcements
- Optional
- Recap of Collections
- **Dumb Data Structures**
- Collections

Dumb Data Structures

We're going to create our own versions of these classes so we can dig into how they all relate to each other!

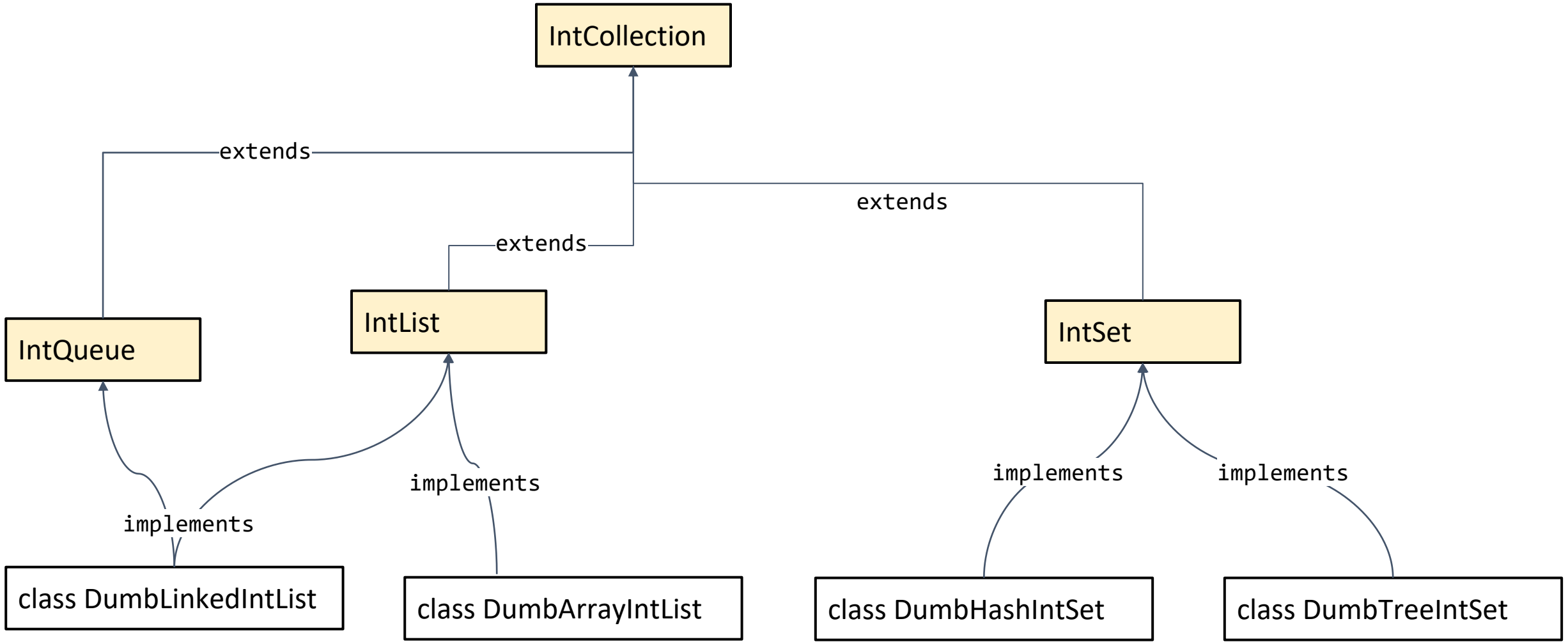
BUT they're going to be real dumb.

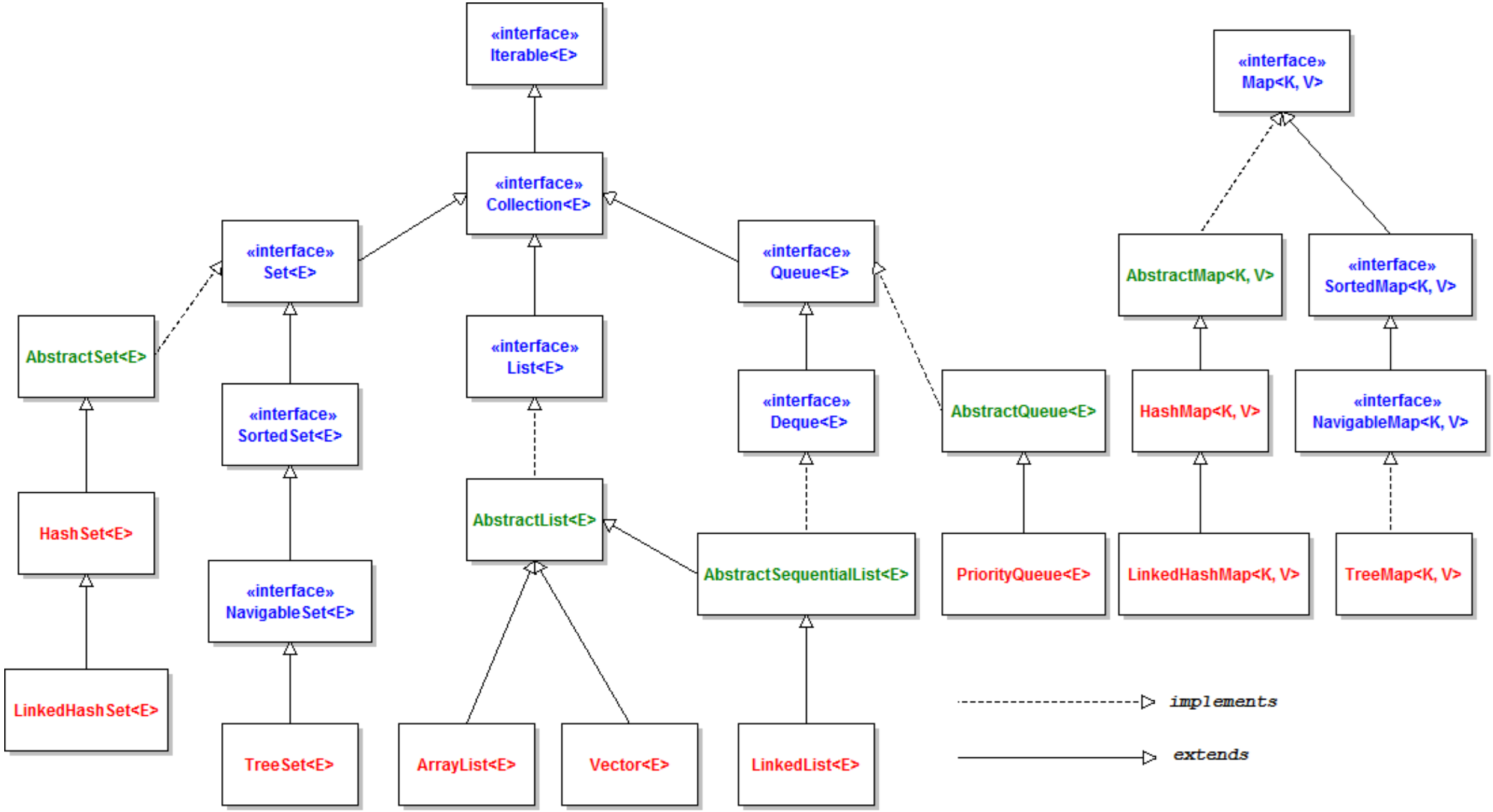
If you want to get a sense of how they're *actually* implemented, go take CSE 123!

Lecture Outline

- Announcements
- Optional
- Recap of Collections
- Dumb Data Structures
- **Collections** 

IntCollection Relationships




«interface» Iterable<E>
«interface» Collection<E>
«interface» Map<K, V>
«interface» Set<E>
«interface» Queue<E>
AbstractMap<K, V>
«interface» SortedMap<K, V>
AbstractSet<E>
«interface» List<E>
«interface» Deque<E>
AbstractQueue<E>
HashMap<K, V>
«interface» NavigableMap<K, V>
«interface» SortedSet<E>
AbstractList<E>
HashSet<E>
AbstractSequentialList<E>
PriorityQueue<E>
LinkedHashMap<K, V>
TreeMap<K, V>
«interface» NavigableSet<E>
LinkedHashSet<E>
TreeSet<E>
ArrayList<E>
Vector<E>
LinkedList<E>
implements
extends