LEC 15

CSE 122

Collections

BEFORE WE START

Talk to your neighbors:

coffee or tea?

Instructors Tristan Huber & Hunter Schafer

TAs

Ambika
Andrew
Audrey
Autumn
Ayush
Ben
Colton
Di
Eesha
Elizabeth

Evelyn
Jacob
Jaylyn
Jin
Joe
Kevin
Leon
Megana
Melissa
Mia

Poojitha
Rishi
Rucha
Shivani
Shreya
Steven
Suhani
Yijia
Ziao

Questions during Class?

Raise hand or send here

sli.do #cse122

Lecture Outline

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

Announcements

- P3 released today!
  - OOP, Interfaces

- Quiz 2 - @ home next Tuesday (5/23)
  - No Section on 5/23
  - Logistics shared on Ed
  - Links & access details will be sent on Monday

Lecture Outline

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

(PCM) 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>`

(PCM) 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!

(PCM) 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)

(PCM) 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!

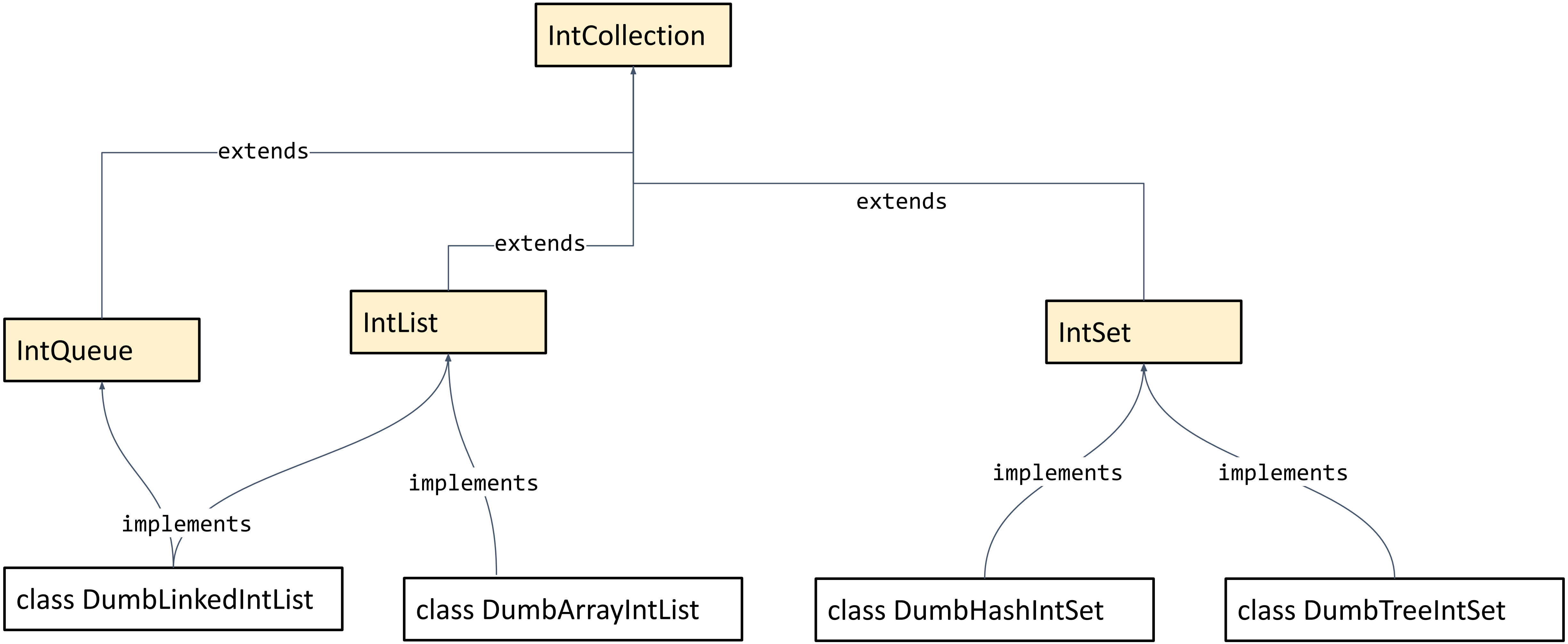
DumbArrayIntList

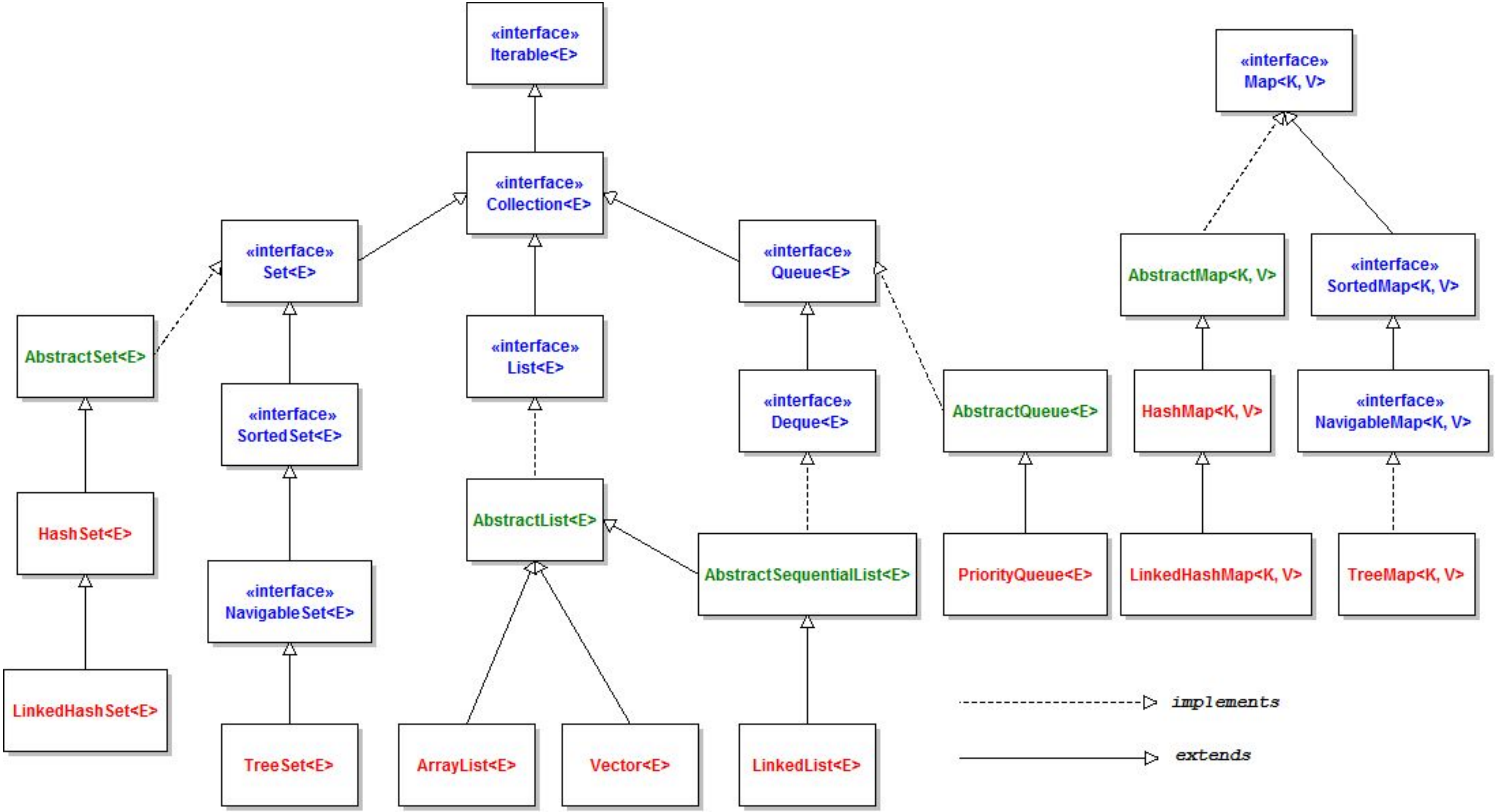
```
DumbArrayIntList()

add(int value)

add(int index, int value)

contains(int value)

isEmpty()

get(int index)


set(int index, int value)

remove(int index)
```

Lecture Outline

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

IntCollection Relationships




«interface»
Iterable<E>
«interface»
Collection<E>
«interface»
Set<E>
«interface»
List<E>
«interface»
Queue<E>
«interface»
Map<K, V>
AbstractSet<E>
«interface»
SortedSet<E>
«interface»
Deque<E>
AbstractQueue<E>
AbstractMap<K, V>
«interface»
SortedMap<K, V>
HashMap<K, V>
«interface»
NavigableMap<K, V>
HashSet<E>
AbstractList<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