Discussion 5: Java Collections library
ADTs, data structures, interfaces, classes

- ADT operations can be declared and specified in a Java interface
Java's **List<E>** interface

- Interfaces for many ADTs in `java.util` package
  - Known as [Java Collections Framework](https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html)
- **Generic** interfaces – type parameter `E` for type of elements
- List operations:
  - `size()` // not "length"
  - `get(i)` // returns an `E`
  - `set(i, e)` // `e` has type `E`
  - `add(i, e)`
  - `remove(i)`
  - `contains(e)`
Example: `List<Course>`

- Can replace A2’s `arrays` of Students and Courses with `Lists`
ADTs, data structures, interfaces, classes

• **ADT** operations can be declared and specified in a Java interface

• A Java **class** implementing such an interface will use **data structures** to implement that functionality

• Multiple classes can implement the same interface using different data structures
List implementations

- [JavaDoc](#): All Known Implementing Classes
  - `ArrayList<E>`: Uses a resizable array
  - `LinkedList<E>`: Uses a (doubly) linked list
- All support the same core operations
Other collection ADTs

• **Collection<E>**
  • Keeps track of objects that have been added, but does not remember order

• **Set<E>**
  • A collection with no duplicates. Common operation: `contains(e)`

• **SortedSet<E>**
  • Iteration order is guaranteed to be sorted (according to value comparisons)

Data structures for these (binary search trees, hash tables) will be taught later, but as a *client*, you can use them now (*HashSet*, *TreeSet*)
Example: \texttt{Set<Student>}

- Can replace A2's \texttt{StudentSet} by leveraging standard class with a custom parametric type
  - Or could implement \texttt{StudentSet} using a field of type \texttt{Set<Student> - composition}
Iterating over collections

• Common operation for all collections: ability to **enumerate** all elements (order may be unspecified)

• Most convenient: "enhanced for-loop"

```java
Collection<String> c = ...;
for (String s : c) {
    // Use s
}
```

• Uses Iterators under the hood: hasNext() & next()
Enhanced for-loops are translated into while loops

List<String> names = ...; List<String> names = ...
Iterator<String> it = names.iterator();
for (String name : names) {
    while (it.hasNext()) {
        String name = it.next();
    }
}