WEEK SIX

Acknowledgements: Slides created based off material provided by Dr. Michael Raymer and Dr. Travis Doom

ARRAYLIST EXAMPLE

- ArrayLists do not *require* you to specify a data type when you define them
- So why do we?
- Let's take a look

HOW DOES ARRAYLIST WORK WITH DIFFERENT DATA TYPES?

Because of generics!

GENERICS

- Allow classes to work with various data types
- Signified by a single capital letter (usually E for Element)
  - Others include N for Number, T for Type, K for Key, V for Value
- Often accompanied by <>
- Help catch typing errors early
- Control the type a container will hold
- Avoid extra typecasting

GENERIC CLASSES

- If we want our class to work with any data type, we add the *type parameter* to the class header inside <>
  - `public class BingoMachine<E>`
- Represents a class with a collection of elements that are stored and procured randomly
- Can store different kinds of objects because of the generic type E

BINGOMACHINE DESIGN

BingoMachine
-contents : ArrayList <E> -rng : Random
+BingoMachine() +add(item : E) +pickItem() : E +isEmpty() : boolean +clear()

- ASIDE: Why do we contain an ArrayList instead of extending it?
  - Extending means we allow access to *all* of ArrayLists' methods
  - Do we want the user to be able to get any item on their own?
  - By containing ArrayList, we force the user to only be able to access an item via the pickItem() method (a random pick)
- Let's implement it!!!!!!!!!!! :D :D :D

WHAT IF I WANT TO RESTRICT WHAT CAN BE PUT IN MY CLASS?

Bounded type parameters!

BOUNDED TYPE PARAMETERS

- We use the `extends` keyword to denote the class that restricts our generic type
  - `public class BingoMachine <E extends Number>`
- Now we can only store subclasses of Number in our BingoMachine
- The `extends` keyword is not only used for regular classes but abstract classes and interfaces too
  - `public class BingoMachine <E extends Comparable<E>>`
- Says that the type E must be comparable to itself
- In other words, forces the class E to implement the compareTo method
  - `public boolean compareTo(E otherItem)`

PUTTING EVERYTHING TOGETHER

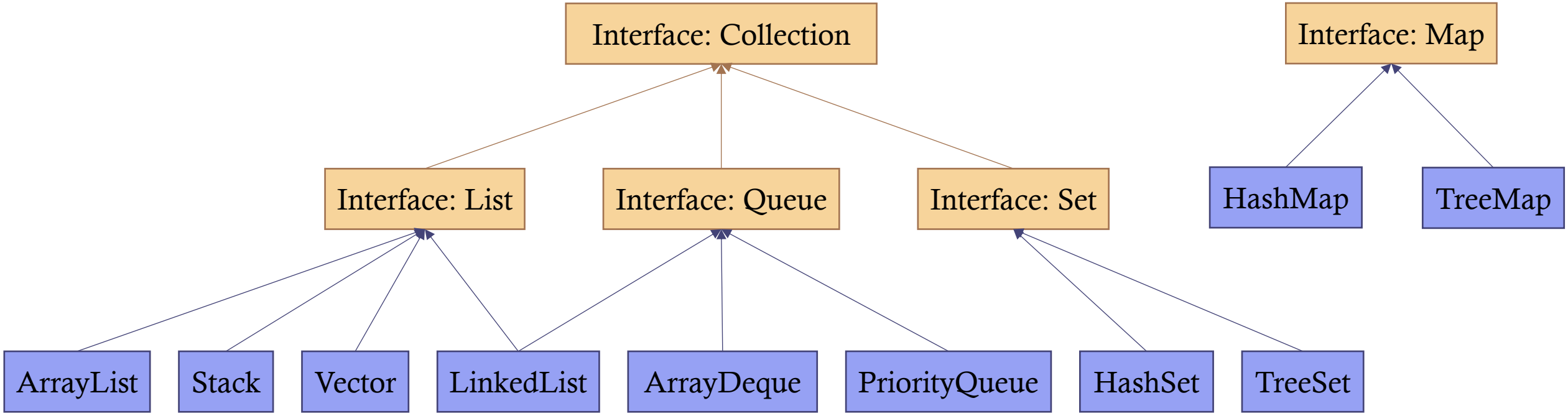
- What does this mean?
- `public class SortedList<E extends Comparable<E>> extends ArrayList<E>`
- SortedList IS-A ArrayList of generic types where the types are comparable to themselves

JAVA COLLECTION INTERFACE

- Stores a collection of elements
- No get() because not all collections have an order

Modifier and Type	Method and Description
boolean	add(E e) Appends the specified element to the end of this list.
boolean	addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection (optional operation).
void	clear() Removes all of the elements from this list.
boolean	contains(Object o) Returns true if this list contains the specified element.
boolean	isEmpty() Returns true if this list contains no elements.
E	remove(**Object o**) Removes the element at the specified position in this list.
int	size() Returns the number of elements in this list.
Object[]	**toArray()** Returns an array containing all of the elements in this collection.

JAVA COLLECTION INTERFACE



JAVA COLLECTIONS

- Java class with a *very* similar name to the interface
- Contains a collection of useful static methods for collections
  - `Collections.sort(List<T> list)`
  - `Collections.max(Collection<? extends T> coll)`
  - `Collections.min(Collection<? extends T> coll)`
  - `Collections.reverse(List<?> list)`
  - `Collections.shuffle(List<?> list)`
  - `Collections.swap(List<?> list, int i, int j)`

MORE ON GENERICS

- Suppose my BingoMachine class is defined as follows
  - `public class BingoMachine <E extends Number>`
- Now suppose I want to make a method that lets me add an entire ArrayList to my object
  - `public void addAll(ArrayList<E> listToAdd)`
- Would this work with an ArrayList of Number?
  - Yes!
- How about an ArrayList of Double?
  - Double is a child of the Number class
  - BUT, ArrayList<Double> is *not* a subclass of ArrayList<Number>

WILDCARDS!

- Represented by a question mark (?)
- By itself, represents an unknown type (any type)
  - Useful for methods that only require functionality of the `Object` type
  - Or if the code uses methods not dependent on the type parameter
- Can be upper-bounded by adding the `extends` keyword
  - `public void addAll(ArrayList<? extends E> listToAdd)`
  - Accepts any ArrayList of type E
  - AND any ArrayList of a type that is a child of E

LOWER-BOUNDED WILDCARDS

- Suppose I want to be able to add everything from my BingoMachine to an ArrayList
  - `public void addToOtherList(ArrayList<E> otherList)`
- Would this work with an ArrayList of Number?
  - Yes!
- What about an ArrayList of Object?
  - Number is a subclass of Object
  - BUT ArrayList<Number> is not a subclass of ArrayList<Object>
- Wildcards can be lower-bounded by adding the `super` keyword
  - `public void addToOtherList(ArrayList<? super E> otherList)`
  - Accepts type E or any parent of type E

GENERIC METHODS

- Generally used for static methods
- Defines the generic type parameter at the method scope rather than the class
- Placed before the return type and after the static keyword
- Can then be used throughout the method (parameter type, return type, inner variable type)
- Examples:
  - `public static <T> void printArray(T[] array)`
  - `public static <T> T findMaxValue(T[] array)`

MORE ON GENERIC METHODS

- If I have:
  - `public static <T> T doStuff(T param1, int param2)`
- Can I do this?
  - `String result = doStuff("helloWorld", 7);`
  - Yes!
- How about this?
  - `String result = doStuff(6, 7);`
  - No! The type for T is determined by the parameter

TYPE ERASURE

- Be aware that behind the scenes, Java still creates your generic class as a collection of Objects and typecasts everything to E
- This is type erasure
- Can cause issues when you are creating new objects of type E
- Examples:
  - ```
    public class Foo<E> {
        T[] bar1 = new T[10]; // Generic Array Creation ERROR
        T[] bar2 = new Object[10]; // Object cannot be converted to T ERROR
        T[] bar3 = (T []) new Object[10]; // okay, as T IS-A Object
    }
    ```
    ```