CS 240: Java Collections – Overview Transcript

This video shows a split screen of Professor Rodham on the right and a PowerPoint screen on the left. Any text displayed or action performed that is not verbalized will be included in italics as visual descriptions.

[00:00:00] **KEN RODHAM:** In this video, we're going to talk about Java collections.

[00:00:04] Java collections are the data structures that are built into the Java language, and as you'd expect, most languages have a library of data structures built-in.

[00:00:14] In C++, they call those containers.

[00:00:17] In Java, they call those collections, but it's pretty much the same thing.

[00:00:21] One of the restrictions on Java collections that you should be aware of is that you can only store objects in the Java collections classes.

[00:00:31] The lists, and the sets, and the maps, and all the things that you can use; they can only store objects.

[00:00:39] Primitive values like integers, floats, Booleans, and so forth cannot be stored in the Java collections.

[00:00:48] This is one place where the wrapper classes that Java provides are important.

[00:00:52] For example, if you wanted to store a list of integers, what you would need to do is store a list of integer objects, Not in primitives.

[00:01:03] Whatever primitive value you might want to store, you'll need to store the wrapper objects instead of the primitive values.

[00:01:13] Other than that, the Java collections library, in my opinion, is very well-designed, and so it's a really good example of library design.

[00:01:21] We're going to just cover the basics.

[00:01:24] We're not going to belabor it too much because you're familiar with a lot of the concepts here, but we do want to at least provide an overview so that you'll know what's available.

[00:01:35] Then the purpose of the Evil Hangman project, which is the next one, the purpose of that lab is to give you experience using the Java collections in a program.

[00:01:48] All right, let's do a brief overview here.

[00:01:52] This slide shows the inheritance hierarchy, the different types, the major types that are in the Java collections library.

[00:02:05] At a high level, there's three different kinds of objects in this library.

[00:02:09] The first is the collections themselves.

[00:02:11] The list classes, the set classes, the queue classes, etc.

[00:02:17] Those are part of the library.

[00:02:21] The next part of the library that's important to understand is iterators.

[00:02:26] The notion of an iterator is, you learn in CS 235, an iterator is an object that lets you enumerate or iterate over the values inside a collection.

[00:02:40] If I wanted to process all the values in a list, for example, I could use an iterator to write a loop to process all the values in a list.

[00:02:48] We'll talk about what iterators look like in Java.

[00:02:52] The other part of the Java collections library is they actually provide a number of algorithms that are commonly used in writing programs.

[00:03:02] As programmers, there's certain algorithms that we use fairly often, like sorting searching, randomization, and things like that.

[00:03:12] Java does provide a set of basic algorithms that you can use in your code.

[00:03:19] These three things make up the Java collections library.

[00:03:24] Now we're going to start by talking about the collection classes, which is what's depicted on the slide.

[00:03:30] You can see, at the root of the inheritance hierarchy, they have this interface called Collection.

[00:03:39] In Java, if you go to the documentation, which we're going to do now. *(Rodham opens the Java documentation website in a browser window.)*

[00:03:48] Go over to the Java documentation.

[00:03:51] We're going to go to the package called java.util. U-T-I-L.

[00:03:57] This is where you'll find the data structures in Java.

[00:04:05] If you look at this java.util package, you'll find lots of interfaces and classes that implement on the collections.

[00:04:15] One of them is called Collection.

[00:04:20] There's an interface called Collection.

[00:04:23] Most of the collections inherit from Collection because the Collection interface defines some methods that are universally supported by almost all data structures, at least everything except the maps.

[00:04:37] If you go to the Collection interface, you'll find methods like add.

[00:04:42] That would be like an insert method on a collection.

[00:04:45] You have a clear method which would empty out the collection.

[00:04:48] You have a contains method, which is like a find method.

[00:04:51] You can pass in an object and ask, is this value in the collection? Is empty.

[00:04:59] Iterators. A very important method on a collection.

[00:05:05] If you want to enumerate all the values that are inside the collection, you call the iterator method and it returns an iterator object to you.

[00:05:13] There's also an interface called Iterator.

[00:05:15] If you look at the Iterator interface, it has a couple of important methods here.

[00:05:21] One is hasNext.

[00:05:23] That method tells you if there are any values remaining in the iteration.

[00:05:28] It returns false at the end when there's no more values to return.

[00:05:31] If hasNext returns true, you can call the next method to get the next value out of the collection.

[00:05:37] Using hasNext and next, you can just write a simple loop that enumerates over a collection.

[00:05:44] The way you get an iterator from a collection is you call the iterator method.

[00:05:50] That's different than in C++. In C++, they have iterators that look like pointers.

[00:05:55] They overload all the pointer operators like asterisk and the arrow, and that's the C++ way of doing iterators.

[00:06:06] Java uses a different approach.

[00:06:07] They just have an Iterator interface with the hasNext and the next methods.

[00:06:11] On the Collection interface, we also have a remove method, a size method, just a lot of methods that every collection would support.

[00:06:22] That's the Collection interface. *(Rodham returns to the PowerPoint slides.)*

[00:06:26] Subclasses of Collection are list, set, and queue.

[00:06:31] We'll talk about those a little more detail in a minute.

[00:06:33] Each of these interfaces add additional methods that are only supported by those specific data structures.

[00:06:38] In CS 235, you learned about lists, sets, queues, maps, and all those things.

[00:06:46] Really, these are just the Java classes that implement those datatypes.

[00:06:51] Then there's some other datatypes here that are maybe less common but still important, like a double-ended queue, the deque, sorted set, so forth.

[00:07:00] The other major datatype in the Java collections library is the map.

[00:07:05] A map is also a collection, but a map is different than other collections because a map is basically a set of key-value pairs.

[00:07:16] That makes it fundamentally different from the other collections.

[00:07:19] That's why it's got its own interface here.

[00:07:22] Then there's different implementations of map.

[00:07:25] These are the abstract data types that are defined by the Java library.

[00:07:29] What I haven't shown you is the concrete classes that actually implement these interfaces and that's what you'd be most interested in your programs.

[00:07:37] Let's talk about some of the concrete classes that implement these datatypes.

[00:07:42] First we have the List interface.

[00:07:46] The List interface has all the methods that we just talked about on the Collection interface like add and remove, an iterator, and those kinds of things.

[00:07:55] The methods that are added by the List interface are the get and set methods because what distinguishes lists is that they're an ordered collection.

[00:08:06] There is an order. There is a first element, the second element, the third element, and so forth.

[00:08:10] With a list, you can access the elements by their index, and you can also set the elements by their index.

[00:08:18] Those are the methods that are unique to the list interface.

[00:08:23] The most common class that we would use for lists in Java would be the ArrayList.

[00:08:30] That's equivalent to the vector class in C++.

[00:08:34] ArrayList is usually the best default list if you need to pick a class for it, but you can also use a LinkedList if you want to have that.

[00:08:46] A LinkedLists is a doubly-linked list in this case.

[00:08:49] Essentially, you have a choice between an array-based list and a doubly-linked list.

[00:08:54] Those two implementations have their strengths and weaknesses, as you've learned about in CS 235.

[00:09:00] For example, in an ArrayList, it's very fast to do random access on the list.

[00:09:06] You can access any element of the list in constant time because it's really just an array access.

[00:09:12] A LinkedList is very effective when you're inserting or deleting elements in the list, either at the beginning or in the middle of the list somewhere.

[00:09:23] It is perhaps more efficient than an ArrayList for doing insertions, and deletions.

[00:09:29] Of course, an ArrayList can do inserts and deletes at the end very efficiently, but it's only at the beginning, and in the middle that it's slower.

[00:09:37] You just have to pick the list implementation that seems to fit the data access patterns of your program and pick the one that you think would be more efficient.

[00:09:47] Lists also have a ListIterator, which is a little more powerful than the iterator I just showed you.

[00:09:54] With lists, the iterators can move forward and backward in the list.

[00:09:59] They have a hasNext method as well as a hasPrevious method.

[00:10:07] Because a list is ordered, you can iterate in both directions: forward and backward.

[00:10:12] The ListIterator is a little bit more powerful than the generic iterator.

[00:10:18] That's how lists look in Java.

[00:10:22] Let's talk about sets next.

[00:10:24] As you know from your prior training, a set is a collection of values that are unique.

[00:10:31] Duplicates are not allowed in a set.

[00:10:35] If you add the same value to a set multiple times, it'll still only be in there once.

[00:10:39] Also, sets are unordered or unsorted.

[00:10:44] That's what distinguishes a set: no duplicates, no order.

[00:10:51] Sets support the dictionary operations that we've discussed previously.

[00:10:58] There's an add method, there's a contains method, and there's a remove method.

[00:11:03] All of those methods actually come from the Collection interface itself.

[00:11:08] These were all inherited. But set does add the extra constraints that duplicates are not allowed, and that there's no order.

[00:11:16] In Java, there's basically two different set implementations you can choose from.

[00:11:22] One is called HashSet, which uses a hash table underneath to implement the set.

[00:11:26] The other one is a TreeSet, which uses a balanced binary search tree underneath to implement the set.

[00:11:35] Again, you can pick the implementation that best meets your needs.

[00:11:41] As we talked about before, hash tables are very fast, but they also take more memory.

[00:11:47] If you want speed and you don't mind spending a little bit of extra memory, then a HashSet's probably a good choice.

[00:11:57] TreeSet would be more memory efficient but somewhat slower.

[00:12:00] TreeSets, of course, use binary search trees, which are Big O of log(N) in performance, whereas HashSets are amortized constant time.

[00:12:13] Just pick the one that meet your needs the best.

[00:12:16] Of course, if you're going to use a HashSet, you have to think about overriding the hash code and the equals methods on your class.

[00:12:22] If you're going to use a TreeSet, you have to implement the comparable interface on your class.

[00:12:29] I'll show you how to do that later in this video.

[00:12:35] Those are your basic choices for sets in Java.

[00:12:42] Queues would be the next data type.

[00:12:46] A queue is defined in Java as a collection designed for holding elements prior to processing which is a really long sentence that says very little.

[00:12:59] Every collection is designed to hold elements prior to processing.

[00:13:03] But I admire their attempt there to define a queue.

[00:13:07] As you learned about in CS 235, there's different queues.

[00:13:12] There's FIFO queues (first in, first out) where you insert elements at the end of the queue and they come out the front of the queue.

[00:13:23] Basically, the element that's been in the queue the longest is the next one that comes out.

[00:13:28] We have LIFO queues (last in, first out) queues, which we usually call stack.

[00:13:34] We also have PriorityQueues, which is a queue where you insert elements into the queue, but when you take the elements out, they come out in priority order.

[00:13:45] There's some notion of some elements are more important than other elements and so the highest priority elements come out first.

[00:13:52] If you think about it, FIFO queues, LIFO queues, and priority queues, they're really the same method interface.

[00:14:00] They have the same methods on them, it just depends on which end of the queue things are coming out of.

[00:14:08] For a FIFO queue, elements come out of the front.

[00:14:13] For a stack, the most recently inserted element comes out the back.

[00:14:19] For a priority queue, they just come out in priority order.

[00:14:24] The methods on a queue in Java would be add.

[00:14:30] That's how you insert a value into the queue. You can peek.

[00:14:34] You can call the peek method to look at the next value that will come out of the queue.

[00:14:39] But peek will leave it in there, it won't take it out.

[00:14:41] It'll just tell you who's going to come out next.

[00:14:43] Then the remove method would actually remove the next value from the queue and return it.

[00:14:47] Depending on which data structure you're looking for, which queue you want, you would use different classes.

[00:14:55] In Java, they have an ArrayDeque, which is an array-based implementation of a queue.

[00:15:01] ArrayDeque would be used for a FIFO queue.

[00:15:07] You can also use it for a LIFO queue as a stack.

[00:15:11] ArrayDeque is really useful for anything but a priority queue.

[00:15:16] You can also use the LinkedList in the same fashion.

[00:15:20] LinkedLists can be used to implement the FIFO queues as well as LIFO queues.

[00:15:27] Usually it's...on which one you use there.

[00:15:32] Then we do have a PriorityQueue class in Java, which is a specialized implementation of a priority queue.

[00:15:37] It's based on a data structure called a binary heap, which you may have learned about in your data structures class.

[00:15:45] Queues are pretty simple.

[00:15:47] Use ArrayDeque, LinkedList, or PriorityQueue based on your needs.

[00:15:53] Another data type in Java's library is the double-ended queue or the deque.

[00:15:59] This word isn't pronounced DQ.

[00:16:01] This is deque, double-ended queue.

[00:16:05] A double-ended queue is a queue that you can insert and remove elements at both ends of the queue efficiently.

[00:16:15] For the other queues that we just talked about, you only insert at one end and you only remove elements from one end, but in a double-ended queue, you can insert values and remove values at both ends of the queue.

[00:16:29] The method interface on a double-ended queue is a little bit different.

[00:16:32] It's got addFirst, addLast.

[00:16:35] You can see you can add at both ends of the queue.

[00:16:37] You can peekFirst, peekLast, removeFirst, removeLast.

[00:16:42] A double-ended queue is guaranteed to be efficient for all of those operations.

[00:16:48] The classes that implement the deque interface are the same as the ones that implemented the queue interface.

[00:16:55] The ArrayDeque and the LinkedList can be used for a double-ended queue, as well as FIFO and LIFO queues.

[00:17:07] Java does have a Stack class, which is deprecated.

[00:17:13] In programming languages, they deprecate various library features when they don't want you to use them anymore.

[00:17:21] The Java Stack class was part of the original Java language, but it had a regrettable design and they eventually decided to deprecate it, which means you shouldn't use it.

[00:17:32] It could go away eventually, although it's been 20 plus years since they deprecated it and it's still hasn't gone away, so I doubt it's ever going to go away actually.

[00:17:40] But it does have some efficiency problems.

[00:17:43] The current advice is if you need to implement a Stack in a program that you would use one of the double-ended queue options, the ArrayDeque or the LinkedList.

[00:18:00] Let's talk about maps.

[00:18:02] As we said earlier, maps are fundamentally different than the other collection types because a map contains a set of key-value pairs, where the keys and the entries are unique.

[00:18:21] It requires a different method interface because you have to deal with key-value pairs.

[00:18:26] The most important methods on the map interface, first of all, would be the put method, which allows you to put a key-value pair into the map.

[00:18:40] Again, the key values are going to be unique.

[00:18:43] If you put an entry into the map that uses the same key as an existing entry, it's going to replace the old entry with the new one.

[00:18:51] You can retrieve the value for a particular key from the map by calling the get method.

[00:18:56] You can ask the map if it contains an entry with a particular key by calling contains.

[00:19:03] You can remove an entry with the remove method, just pass in the key.

[00:19:07] You can also ask a map for a set of all the keys that are in the map and because the keys are unique then, it's appropriate to use a set for that because the elements of a set are also unique.

[00:19:20] You can ask a map to return a collection of all the values that are in the map.

[00:19:26] Now that's not going to be a set because the values are not guaranteed to be unique, so that would return a collection rather than a set.

[00:19:34] The last thing you can ask a map for is a set of all the entries or all the key-value pairs in the set.

[00:19:42] This is actually a really useful method if you want to iterate over all the pairs or all the entries in the set.

[00:19:48] Calling entrySet gives you back a set of all those entries that you can then iterate over and very efficiently process every key-value pair in the whole map.

[00:20:00] In Java, there's basically two different implementations of the map interface.

[00:20:06] There's a HashMap, which is based on a hash table implementation, and a TreeMap, which internally uses a balanced binary search tree to implement the map, very similar to sets.

[00:20:20] Again, if you're going to use a HashMap, then whatever your key type is, has to have a hashCode method on it.

[00:20:27] It has to have an equals method on it that work the way you want them to work.

[00:20:32] For a TreeMap, the key type has to implement the comparable interface so that Java can sort the keys and build the tree.

[00:20:40] I'll talk more about that in a few minutes.

[00:20:47] One thing I neglected to mention when we were talking about sets—I'm going to go back to sets for a minute—is we talked about how HashSets are faster than TreeSets, but they also use more memory than TreeSets.

[00:21:04] The other advantage of a TreeSet is that it's actually sorted.

[00:21:08] By definition, a set is unsorted.

[00:21:11] But actually, we know that if we implement a set with a binary search tree, that it is in fact sorted.

[00:21:16] A TreeSet is an example of what's called a sorted set, which doesn't make sense probably to a mathematician to even say sorted set, but we do it anyway.

[00:21:28] For example, if you wanted to create a set such that when you iterate over the elements of the set, the values come out in sorted order, you would want to use a TreeSet.

[00:21:40] A HashSet would not return the values in sorted order.

[00:21:44] It would return them in some random looking order.

[00:21:50] When we talk about maps, you get a similar trade-off.

[00:21:54] HashMaps use more memory, but they're faster.

[00:21:59] But if you iterate over the key-value pairs, they're not sorted in any way.

[00:22:05] Whereas with a TreeMap, it doesn't use as much memory.

[00:22:09] It's slower, but the key-value pairs are sorted.

[00:22:13] If you iterate over them, they'll come out in sorted order.

[00:22:19] Now one thing to know about all these collection types in Java is that they are all implemented so that they work with the for each loop.

[00:22:28] Java has a for each loop, which has a syntax here at the bottom.

The following lines of code are the for each loop syntax:

Set<String> words;

for (String w : words) {…}

End of code.

[00:22:34] This loop says for each string in the words collection (words is a set)—that's one way in Java you can iterate over all the values in a collection.

[00:22:46] All of these collections are written to work with the for each loop.

[00:22:50] That’s typically the way you would iterate over them, with a for each loop.