Java 21: SequencedCollection, SequencedSet and SequencedMap

Tuesday, 20 May 2025 5:38 PM

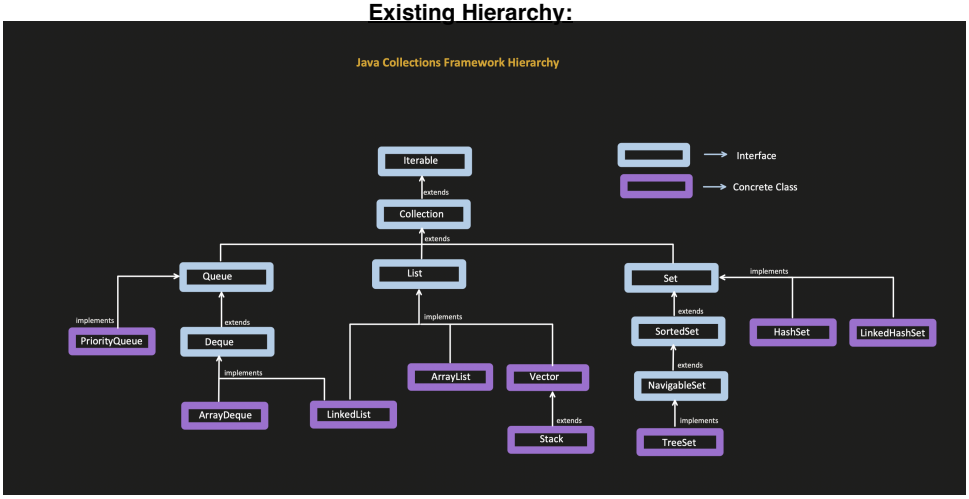
SequencedCollection, SequencedSet and SequencedMap

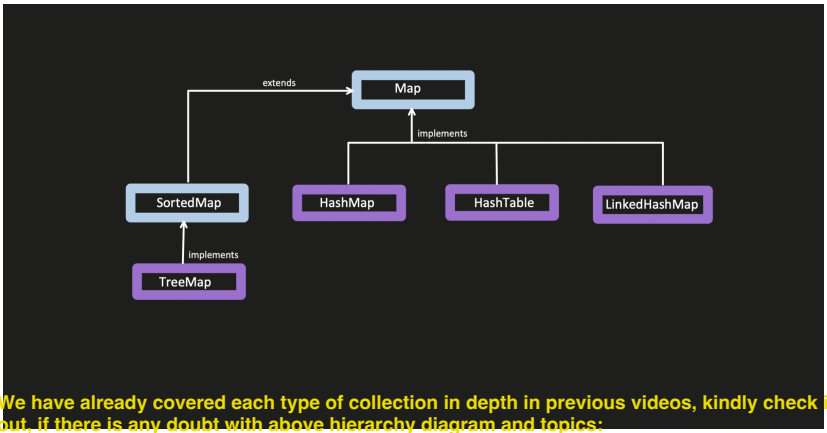
- **These are new interfaces added in existing Java Collection hierarchy.**

So, before we start understanding about these new interfaces,

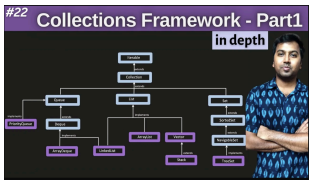
Let's first see, existing Collection hierarchy and where does these new interfaces fit into that:

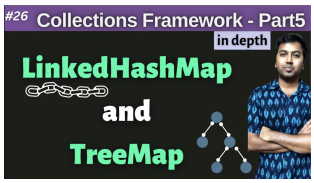
Existing Hierarchy:

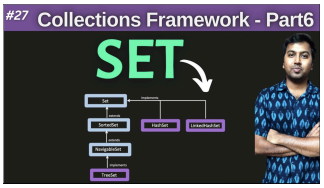




We have already covered each type of collection in depth in previous videos, kindly check it out, if there is any doubt with above hierarchy diagram and topics:

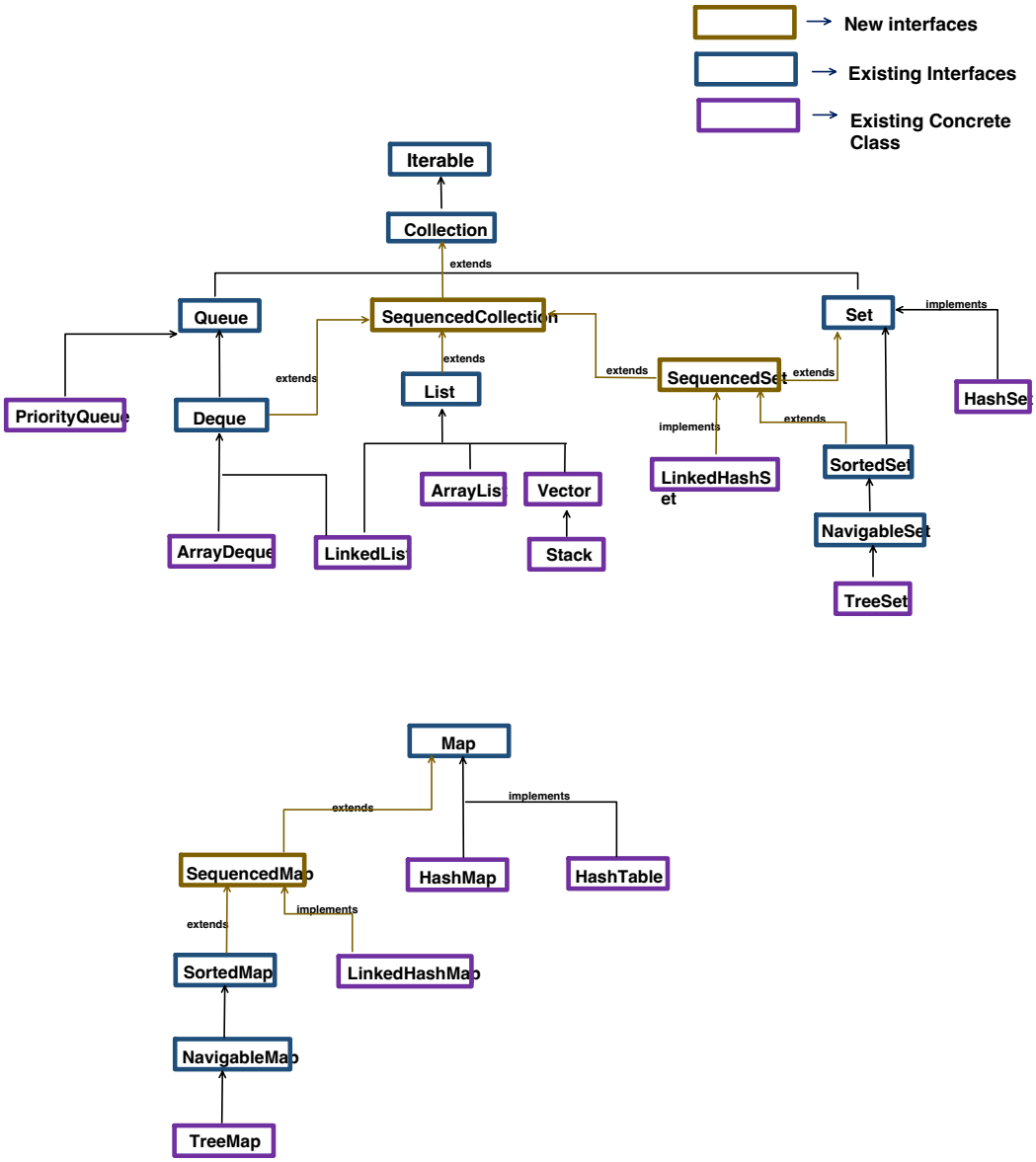






New Collection Hierarchy (Java

21+):



Three questions comes to our mind is:

1. **What's the criteria of putting these interfaces at this specific place? Like I can see it left out Queue and PriorityQueue, why? Why not include those also?**
2. **Why exactly these interfaces required? What is the gap, which it is trying to fill.**
3. **Why SequencedSet is needed, why not SequencedCollection is sufficient for those collections group?**

Once, we understand above points, we will fully understand these SequencedCollection, SequencedSet and SequencedMap

What's the criteria of putting these interfaces at this specific place? Like I can see it left out Queue and PriorityQueue, why? Why not include those also?

Any Collection which follows below conditions, can be termed as Sequenced

- Collection should follow Predictable Iteration :
  Means elements are returned in consistent and well defined order every time we iterate over the collection.
  So if a collection maintains elements in:
    - Insertion order, or
    - Sorted order (e.g., ascending or descending)

  Then we can say, it follows Predictable iteration.

- Collection provide support for Access or Manipulate the First and Last element

- Collection supports Reversible View

Collection	Predictable Iteration	First & Last Element operation	Reversible View	Part of

List	Follows Insertion Order	Access:
		list.get(0);
		list.get(list.size() - 1);
		Add:
		list.add(0, element);
		list.add(element);
		Remove:
		list.remove(0);
		list.remove(list.size() - 1);
			List<String> list = new ArrayList<>
			(List.*of*("a","b","c"));
			Collections.*reverse*(list);
				SequencedCollection

Deque	Follows Insertion Order	Access:
		deque.getFirst();
		deque.getLast();
		Add:
		deque.addFirst(element);
		deque.addLast(element);
		Remove:
		deque.removeFirst();
		deque.removeLast();
			Deque<Integer> revDeque = deque.reversed();	SequencedCollection

Queue	Follows Insertion Order	Follows FIFO
		(insertion at back and remove from front)
		Access:
		queue.peek(); //return head
		Access Last element -
		Add:
		Add element at first -
		queue.add(element); //add to tail
		Remove:
		queue.poll(); //removes head
		Remove last element-
			Not supported directly	N/A

PriorityQueue	Do no follow either Insertion or Sorted order
	Why? bcoz
	• Uses heap.
		Access:
		pq.peek(); //return top priority val
		Access Last element -
		Add:
		Add element at first -
		Add element at last-
			Not supported	N/A

	• Which guarantees the head is either minimum or maximum element, depending on the comparator.
	• But the iteration order is not properly sorted or consistent
		As addition at first or last happens based on priority
		
		Remove:
		pq.poll(); //removes top priority val
		Remove last element-		

HashSet	Do no follow either Insertion or Sorted order	Since order is not supported, access first or last element is not supported too.	Not supported	N/A

LinkedHashSet	Follows Insertion Order	Access:
		Requires manual iteration and then we can access both first and last element.
		ex:
		linkedHS.iterator().next(); //first element
		
		Access First Element:
		Access Last Element:
		
		Add:
		
		Add element at First:
		(Yellow because, it uses doubly LinkedList internally, so its possible to add element at First, but method is not present)
		
		linkedHS.add("A"); — Adds to the end
		
		Remove:
		Requires manual iteration and then we can remove both first and last element.
		ex:
		Iterator<E> it = set.iterator();
		it.next();
		it.remove();
		
		Remove First Element:
		Remove Last Element:
			As LinkedHashSet, uses doubly linked list internally, so its possible to get reverse view of the collection.
			
			But there is no API or method exposed before Java21.
				SequencedSet,
				
				Reason why LinkedHashSet, is put inside SequencedSet and not directly under SequencedCollection because, SequencedSet adds an additional contract i.e. NO DUPLICATES.

SortedSet	Follows Sorted Order	SortedSet<Integer> sortedSet = new TreeSet<>(Set.of(14, 5, 7));
		Access:
		sortedSet .first(); // return 5 (minimum)
		sortedSet .last(); // return 14 (maximum)
		
		Add:
		sortedSet .add(2); // Goes to first if smallest
		sortedSet .add(25); // Goes to last if largest
		
		Remove:
		
		sortedSet .remove(sortedSet .first());
		sortedSet .remove(sortedSet .last());
			NavigableSet<Integer> sortedSet = new TreeSet<>(Set.of(14, 5, 7));
			
			Iterator<String> revItr = sortedSet.descendingIterator();
			while(revItr.hasNext()) {
			//iteration logic
			}
				SequencedSet

HashMap	Do no follow	Since order is not supported, access first or last element is not supported too.	Not supported	N/A
HashTable	Do no follow	Since order is not supported, access first or last element is not supported too.	Not supported	N/A
LinkedHashMap	Follows Insertion Order	Access: map.entrySet().iterator().next() Similarly last element can be iterate. Add: Add element at First:	(Reverse view is possible as it uses doubly LinkedList, but method is not present)	SequencedMap

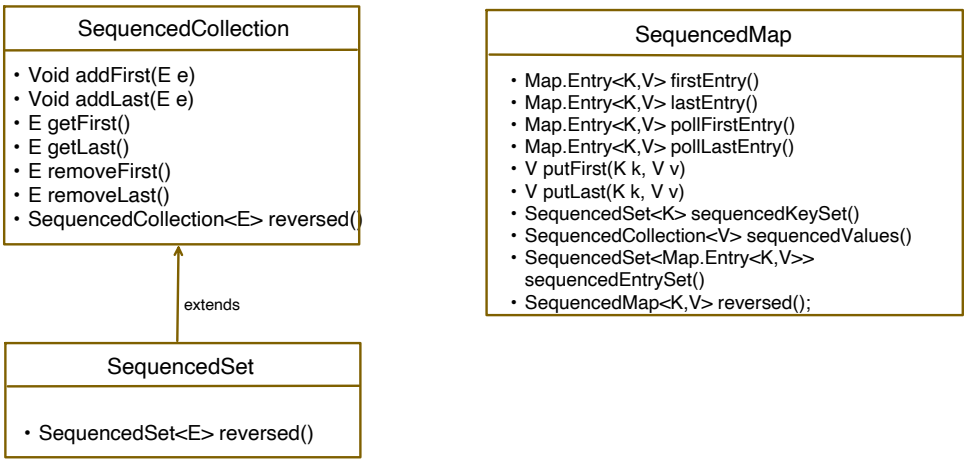
		(Yellow because, it uses doubly LinkedList internally, so its possible to add element at First, but method is not present) map.put(key, value) It adds the entry at the end of the map iteration order. Remove: Requires manual iteration and then we can remove both first and last element. ex: map.entrySet().iterator().remove() Remove First Element: Remove Last Element:		
SortedMap	Follows Sorted Order	Access: firstKey() lastKey() Add: put(K key, V value) Remove: pollFirstEntry() (via NavigableMap) pollLastEntry() (via NavigableMap)	In NavigableMap, we have the method "descendingMap()", we can use it to iterate in reverse direction.	SequencedMap

2. Why exactly these interfaces required? What is the gap, which it is trying to fill.

- If we observe the Collections which are Sequenced. We will find that:
  - Get first
  - Get Last
  - Add first
  - Add last
  - Remove first
  - Remove last
  - Reverse View of the collection

There is no common interface, each collection have their own method. Which is difficult to remember and maintain.

That's one of the gap, which these SequencedCollection, SequencedSet and SequencedMap are trying to fill up.



Now, lets do the same operations using Java 21

Collection	Predictable Iteration	First & Last Element operation	Part of

List	Follows Insertion Order	List<String> list = new ArrayList<>(List.of("B", "C", "D"));
		
		Access:
		list.getFirst(); //B
		list.getLast(); //D
		Add:
		list.addFirst("A"); //A, B, C, D
		list.addLast("Z"); //A, B, C, D, Z
		
		Remove:
		list.removeFirst(); //B, C, D, Z
		list.removeLast(); //B, C, D
		
		Reverse:
		list.reversed(); //D, C, B
			Sequenced Collection

Deque	Follows Insertion Order	Deque<String> deque= new ArrayDeque<>(List.of("B", "C", "D"));
		
		Access:
		deque.getFirst(); //B
		deque.getLast(); //D
		Add:
		deque.addFirst("A"); //A, B, C, D
		deque.addLast("Z"); //A, B, C, D, Z
		
		Remove:
		deque.removeFirst(); //B, C, D, Z
		deque.removeLast(); //B, C, D
		
		Reverse:
		deque.reversed(); //D, C, B
			Sequenced Collection

LinkedHashSet	Follows Insertion Order	SequencedSet<String> set= new LinkedHashSet<>(List.of("B", "C", "D"));
		
		Access:
		set.getFirst(); //B
		set.getLast(); //D
		Add:
		set.addFirst("A"); //A, B, C, D
		set.addLast("Z"); //A, B, C, D, Z
		
		//as Set do not contains duplicate, so when we try to insert the duplicate value, it will find the existing value and shift it to the new place.
		set.addFirst("C"); //C, A, B, D, Z
		
		Remove:
		set.removeFirst(); //A, B, D, Z
		set.removeLast(); //A, B, D
		
		Reverse:
		set.reversed(); //D, B, A
			SequencedSet

SortedSet	Follows Sorted Order	SequencedSet<Integer> sortedSet= new TreeSet<>(Set.of(14, 5, 7));
		
		Access:
		sortedSet.getFirst(); //5
		sortedSet.getLast(); //14
		Add:
		//as SortedSet sort the values as when inserted, so addFirst() and addLast() method, do not make sense, that's why this method throws UnsupportedOperationException.
		
		sortedSet.addFirst(2); //UnsupportedOperationException
		sortedSet.addLast(25); //UnsupportedOperationException
		
		//Instead we can simply use add() method
		
		sortedSet.add(2); //2, 5, 7, 14
		sortedSet.add(25); //2, 5, 7, 14, 25
		
		Remove:
		sortedSet.removeFirst(); //5, 7, 14, 25
		sortedSet.removeLast(); //5, 7, 14
		
		Reverse:
		sortedSet.reversed(); //14, 7, 5
			SequencedSet

	Follows Insertion Order	SequencedMap<Integer, String> map= new LinkedHashMap<>();	

LinkedHashMap		map.put(100, "B");
		map.put(200, "C");
		map.put(300, "D");
		
		Access:
		map.firstEntry(); //100=B
		map.lastEntry(); //300=D
		
		Add:
		map.putFirst(400, "A"); //400=A , 100=B, 200=C, 300=D
		map.putLast(500, "Z"); //400=A , 100=B, 200=C, 300=D, 500=Z
		
		Remove:
		map.pollFirstEntry(); //100=B, 200=C, 300=D, 500=Z
		map.pollLastEntry(); //100=B, 200=C, 300=D
		
		Reverse:
		map.reversed(); //300=D, 200=C, 100=B
			SequencedMap

SortedMap	Follows Sorted Order	SortedMap, sort the order based on keys
		
		SequencedMap<Integer, String> sortedMap= new TreeMap<>();
		sortedMap.put(100, "B");
		sortedMap.put(200, "C");
		sortedMap.put(300, "D");
		
		Access:
		sortedMap.firstEntry(); //100=B
		sortedMap.lastEntry(); //300=D
		
		Add:
		//as SortedMap sort the values as when inserted, so putFirst() and putLast() method, do not make sense, that's why this method throws UnsupportedOperationException.
		
		sortedMap.putFirst(50, "A");
		//UnsupportedOperationException
		sortedMap.putLast(400, "Z");
		//UnsupportedOperationException
		
		//Instead we can simply use put() method
		sortedMap.put(50, "A"); //50=A , 100=B, 200=C, 300=D
		sortedMap.put(400, "Z"); //50=A , 100=B, 200=C, 300=D, 400=Z
		
		Remove:
		sortedMap.pollFirstEntry(); //100=B, 200=C, 300=D, 400=Z
		sortedMap.pollLastEntry(); //100=B, 200=C, 300=D
		
		Reverse:
		sortedMap.reversed(); //300=D, 200=C, 100=B
			SequencedMap