Sequenced Collections (JEP 431)

📦 Introduced in **Java 21**

👤 Stuart Marks

Sequenced collections - The Problem ❌

Many collections **do have order**, but API is inconsistent:

- `List.get(0)` vs `Deque.getFirst()` vs `SortedSet.first()`
- `LinkedHashSet` → no easy way to get last element

Reverse iteration = messy, inconsistent, sometimes impossible

Encounter order lost when wrapping with unmodifiable collections

Sequenced collections - The Goal 🎯

Define **a unified type** for collections with encounter order

Provide **consistent APIs**:

- `getFirst()`, `getLast()`
- `addFirst()`, `addLast()`
- `removeFirst()`, `removeLast()`

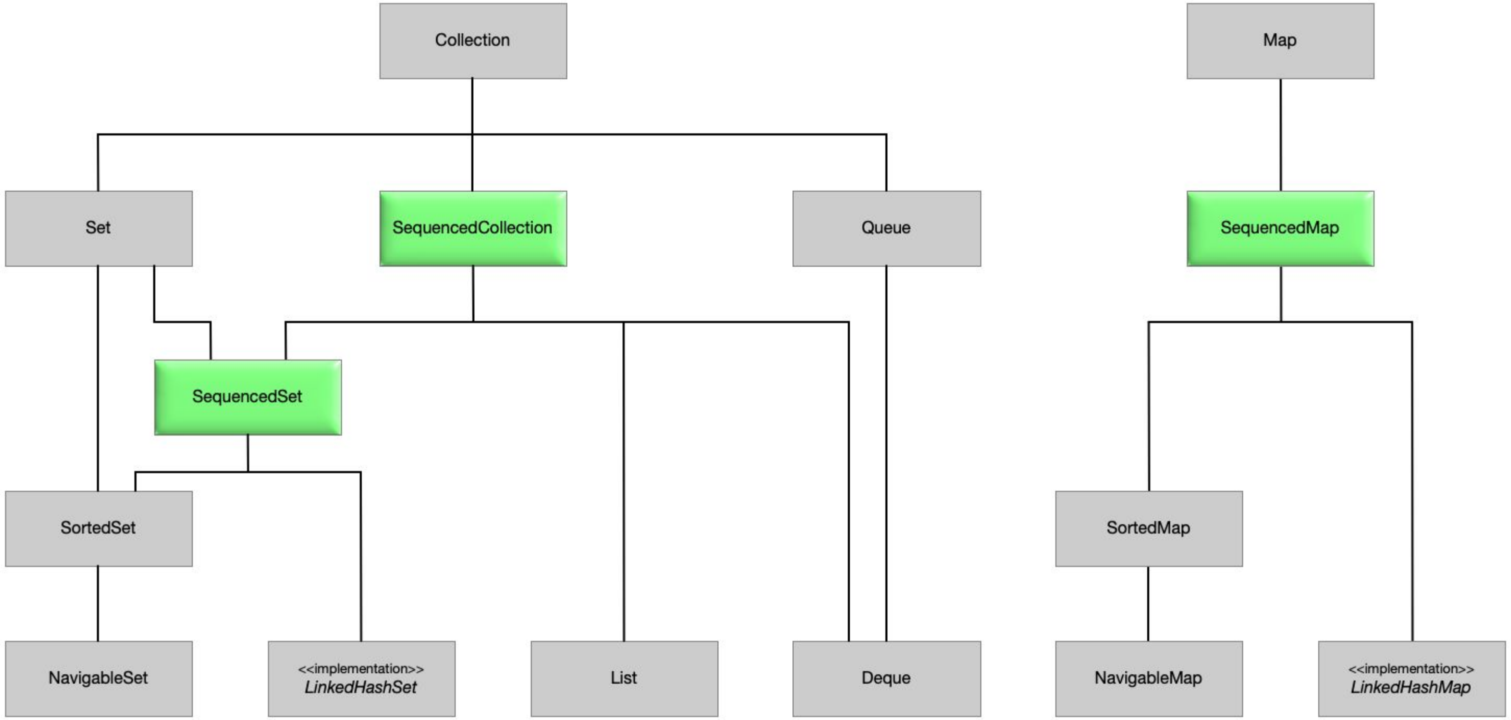
Enable **reverse iteration** easily

Sequenced collections - New Interfaces 🌱

- **SequencedCollection<E>** (extends `Collection<E>`)
- **SequencedSet<E>** (extends `Set<E>, SequencedCollection<E>`)
- **SequencedMap<K,V>** (extends `Map<K,V>`)

✅ Retrofitted into existing hierarchy

Sequenced collections - New Interfaces 🌱



Sequenced collections - Example: SequencedCollection

```
List<String> champions = new ArrayList<>(

  List.of("Josh", "Venkat", "Ixchel")

);


champions.addFirst("Andres");

champions.addLast("Ken");


champions = champions.reversed();


champions.removeFirst();

champions.removeLast();


System.out.println(champions.getFirst());

System.out.println(champions.getLast());
```

https://dev.java/playground/

Sequenced collections - Example: SequencedMap

```
SequencedMap<Integer, String> map = new LinkedHashMap<>();

map.putFirst(1, "First");
map.putLast(2, "Last");

System.out.println(map.firstEntry());  // 1=First
System.out.println(map.lastEntry());   // 2=Last

map = map.reversed();

System.out.println(map.firstEntry());  // 1=Last
System.out.println(map.lastEntry());   // 2=First
```

https://dev.java/playground/

Sequenced collections - Retrofitting 🔄

`List` and `Deque` → now extend **SequencedCollection**

`SortedSet` → now extends **SequencedSet**

`LinkedHashSet` → implements **SequencedSet**

`SortedMap` → now extends **SequencedMap**

`LinkedHashMap` → implements **SequencedMap**

Sequenced collections - Benefits ✅

Uniform, consistent APIs across all ordered collections

No more hacks for **last element** or reverse iteration

Reposition elements in `LinkedHashSet` and `LinkedHashMap`

Better interoperability with `Collections.unmodifiable*()`

Sequenced collections - Design Choices ⚖️

Couldn’t just reuse `List` (too specific, requires index access)

Couldn’t just reuse `Deque` (too cluttered with queue ops)

New dedicated types = cleaner, consistent API

Sequenced collections - Key Takeaways 💡

Encounter order now has **first-class API** in Java

Makes ordered collections **easy to use & extend**

Huge win for framework authors & everyday developers