

Collection Sequences, Operations and Factories



Richard Warburton

Java champion, Author and Programmer

@richardwarburto www.insightfullogic.com

Overview

Factories

How to make
unmodifiable,
immutable, empty or
wrapping collections

Operations

Useful collection
algorithms

Sequenced Collections

A uniform API for
collections with
encounter order



| Unmodifiable Factories (Live Coding)



Factory Method Options (Live Coding)



Factory Methods



```
List<String> list = Collections.emptyList();  
Map<Integer, String> map = Collections.emptyMap();  
Set<Integer> set = Collections.emptySet();
```

Empty Collections

Immutable

Use when you want to pass a no values to a method that takes a collection



```
List<String> list = Collections.singletonList("one");  
Map<Integer, String> map = Collections.singletonMap(1, "one");  
Set<Integer> set = Collections.singleton(1);
```

Singletons

Immutable single value of collection

Use when you want to pass a single value to a method that takes a collection



```
List<String> list =  
    List.of("UK", "USA");  
  
Map<String, Integer> map =  
    Map.of("UK", 67, "USA", 328);  
  
Map<String, Integer> entries =  
    Map.ofEntries(  
        Map.entry("UK", 67),  
        Map.entry("USA", 328));
```

◀ Collection factories
◀ Alternative to collection literals
◀ Runtime immutable – add throws exception
◀ Overloads for performance
◀ Alternative map



Immutable Copies

```
// Modifying countries does not modify immutableCountries
Collection<String> countries = new ArrayList();
countries.add("UK"); countries.add("USA");

List<String> immutableCountries = List.copyOf(countries);

Map<String, Integer> populations = new HashMap<>();
populations.put("UK", 67); populations.put("USA", 328);

Map<String, Integer> immutablePopulations = Map.copyOf(populations);
```



Unmodifiable Views

```
// Modifying countries is the only way to modify countriesView
List<String> countries = new ArrayList<>();
countries.add("UK"); countries.add("USA");

List<String> countriesView = Collections.unmodifiableList(countries);

Map<String, Integer> populations = new HashMap<>();
populations.put("UK", 67); populations.put("USA", 328);

Map<String, Integer> populationsView = Collections.unmodifiableMap(populations);
```



Collection Operations (Live Coding)





Collection Operations



Disjoint

```
var _1to3 = List.of(1, 2, 3);
var _2to4 = List.of(2, 3, 4);
var _4to6 = List.of(4, 5, 6);

System.out.println(Collections.disjoint(_1to3, _4to6)); // true
System.out.println(Collections.disjoint(_1to3, _2to4)); // false
```



```
var letters = "ABCDEFAADSEA".chars().mapToObj(x -> (char)x).toList();  
int count = Collections.frequency(letters, 'A');  
System.out.println(count); // 4
```

Frequency

Returns the number of elements within a collection equal to the object



Addall

```
// Adds multiple elements to a collection

var door = new Product("Wooden Door", 35);
var floorPanel = new Product("Floor Panel", 25);
var window = new Product("Glass Window", 10);

var products = new ArrayList<Product>();
Collections.addAll(products, door, floorPanel, window);
```



```
var door = new Product("Wooden Door", 35);
var floorPanel = new Product("Floor Panel", 25);
var window = new Product("Glass Window", 10);

var products = List.of(door, floorPanel, window);
var max = Collections.max(products, Product.BY_WEIGHT);
System.out.println(max == door);
```

Max and Min

Find the maximum or minimum value in a collection based upon a comparator



```
Collections.fill(products, door);
```

◀ **Fill replaces every element in the collection with the provided parameter**

```
Collections.swap(products, 1, 2);
```

◀ **Swap over two elements in a List by index**

```
Collections.reverse(products);
```

◀ **Reverse the order of elements in a List**



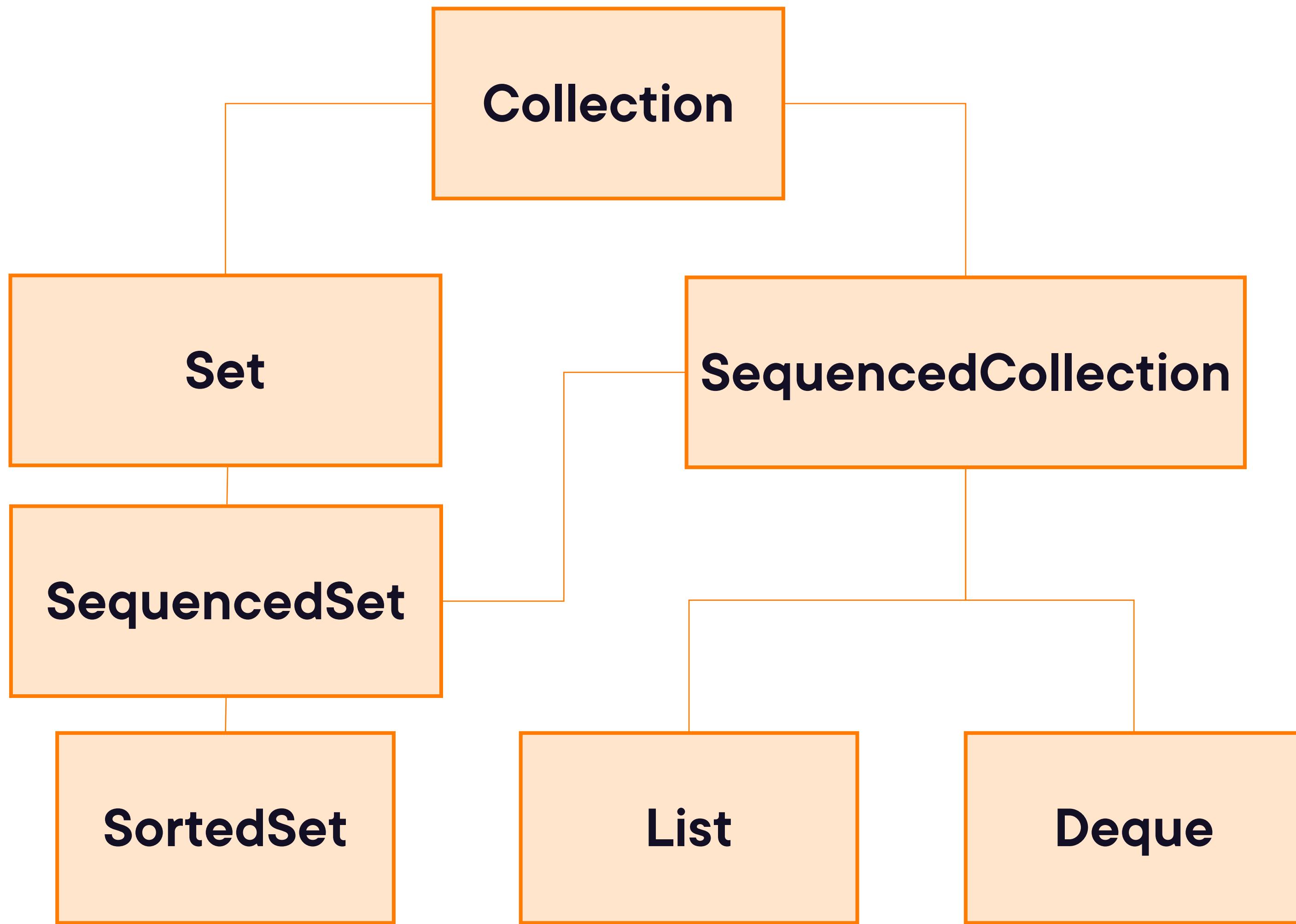
Sequenced Collections (Live coding)

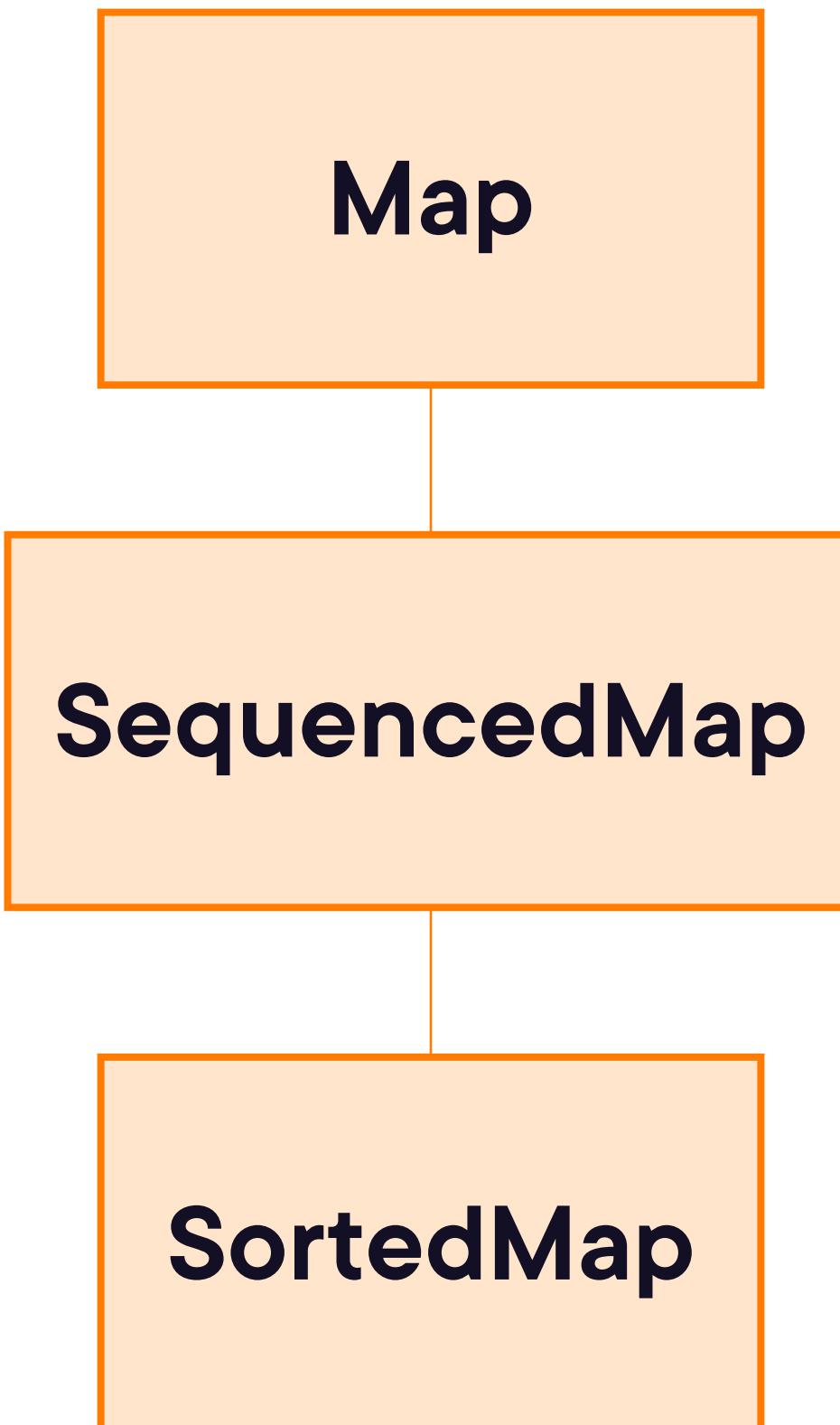




Sequenced Collections







Summary

Collections aren't just about data structures

Common operations ship with the JDK

Immutable + unmodifiable collections reduce scope for bugs

Sequenced Collections add more consistency in JDK 21.



Up Next:

Up Next:

Collections with Uniqueness: Sets

