Lập trình hướng đối tượng

Biên soạn: Nguyễn Thị Tuyết Hải

Email: tuyethai@ptithcm.edu.vn

Chapter 9: Collections

9.1 The Java Collections Framework

9.1.1 Separating Collection Interfaces and Implementation

E.g.,

- Interface tells nothing about how the queue is implemented.

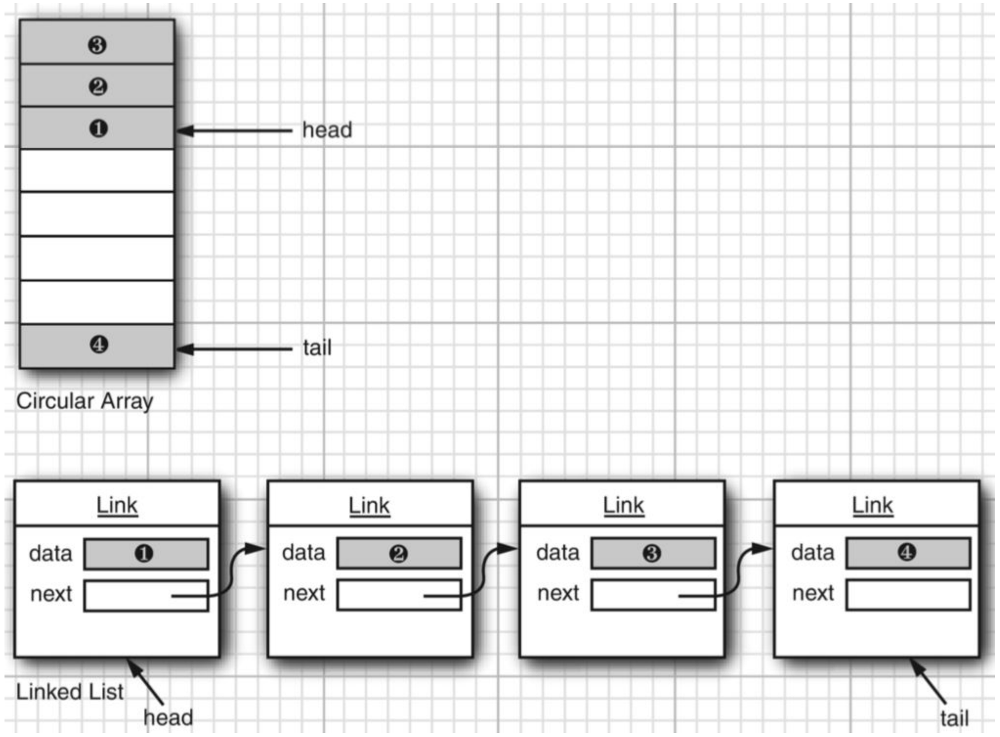
```
public interface Queue<E>{
        void add(E element);
        E remove();
        int size();
}
```

- Implementation:

public class CircularArrayQueue<E> implements Queue<E> …
public class LinkedListQueue<E> implements Queue<E> …

- When use a queue, don't need to know which implementation is actually used once the collection has been constructed.

```
Queue<Customer> expressLane = new CircularArrayQueue<>(100);
expressLane = new LinkedListQueue<>();
```



9.1 The Java Collections Framework

9.1.2 The Collection Interface

Two fundamental methods:

```
    public interface Collection<E> {
        boolean add(E element);
        Iterator<E> iterator();
        ...
    }
```

9.1.3 Iterators

```
public interface Iterator<E> {
    E next();
    boolean hasNext();
    void remove();
    default void forEachRemaining(Consumer<? super E> action);
}
```

9.1 The Java Collections Framework

9.1.3 Iterators

- inspect all elements in a collection:
    - request an iterator
    - keep calling the next method while hasNext returns true
- “for each” loop works with any object that implements the Iterable interface
    - collection interface extends the Iterable interface

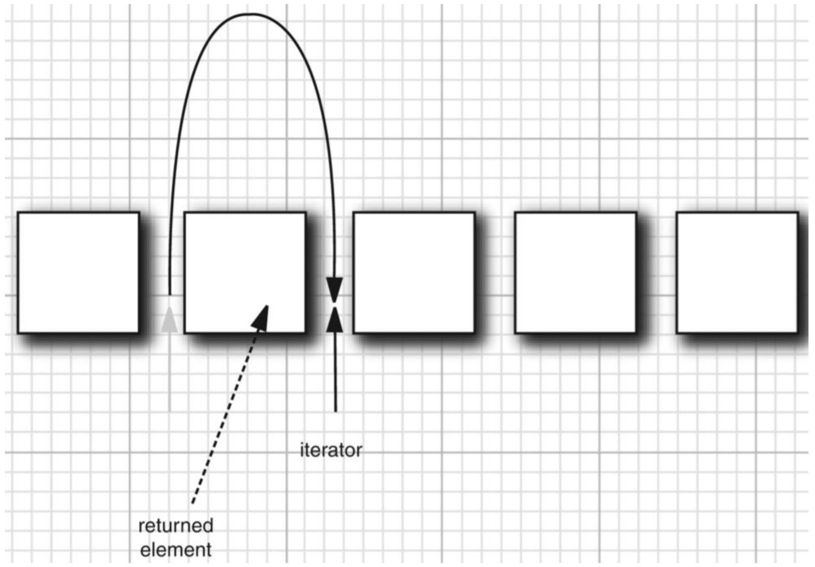
```
Collection<String> c = . . .;
Iterator<String> iter =
c.iterator();
while (iter.hasNext()){
      String element = iter.next();
      // do something with element
}

// OR
for (String element : c) {
      do something with element
}
```

9.1 The Java Collections Framework

9.1.3 Iterators

- Java iterators as being between elements.
- next(), the iterator
  - jumps over the next element,
  - returns a reference to the element that it just passed

```
// remove 1st element
Iterator<String> it = c.iterator();
// skip over the first element
it.next();
// now remove it
it.remove();

//remove two adjacent elements
it.remove();
it.next();
it.remove();
```

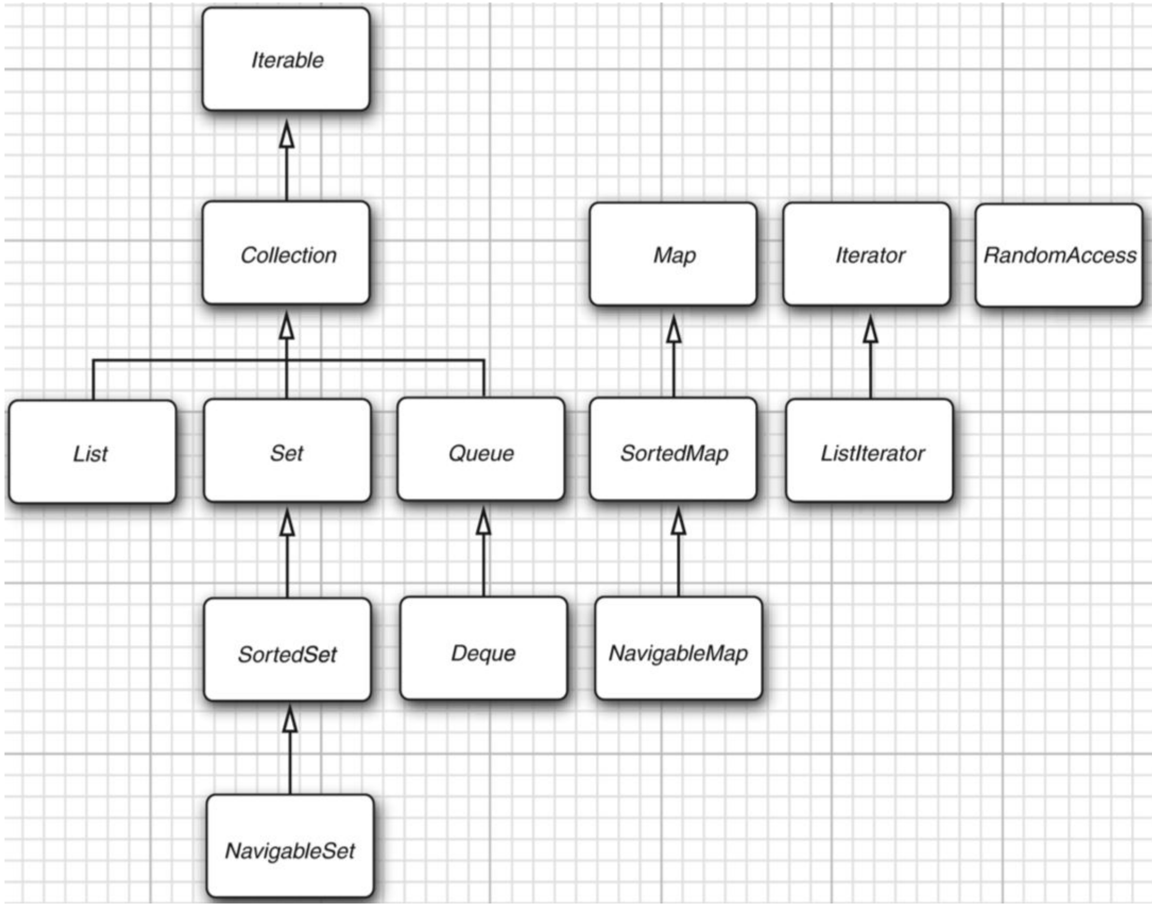


9.1.4 Generic Utility Methods

- Collection and Iterator interfaces are generic
- Collection interface declares quite a few useful methods that all implementing classes must supply.
- Class AbstractCollection leaves the fundamental methods size and iterator abstract

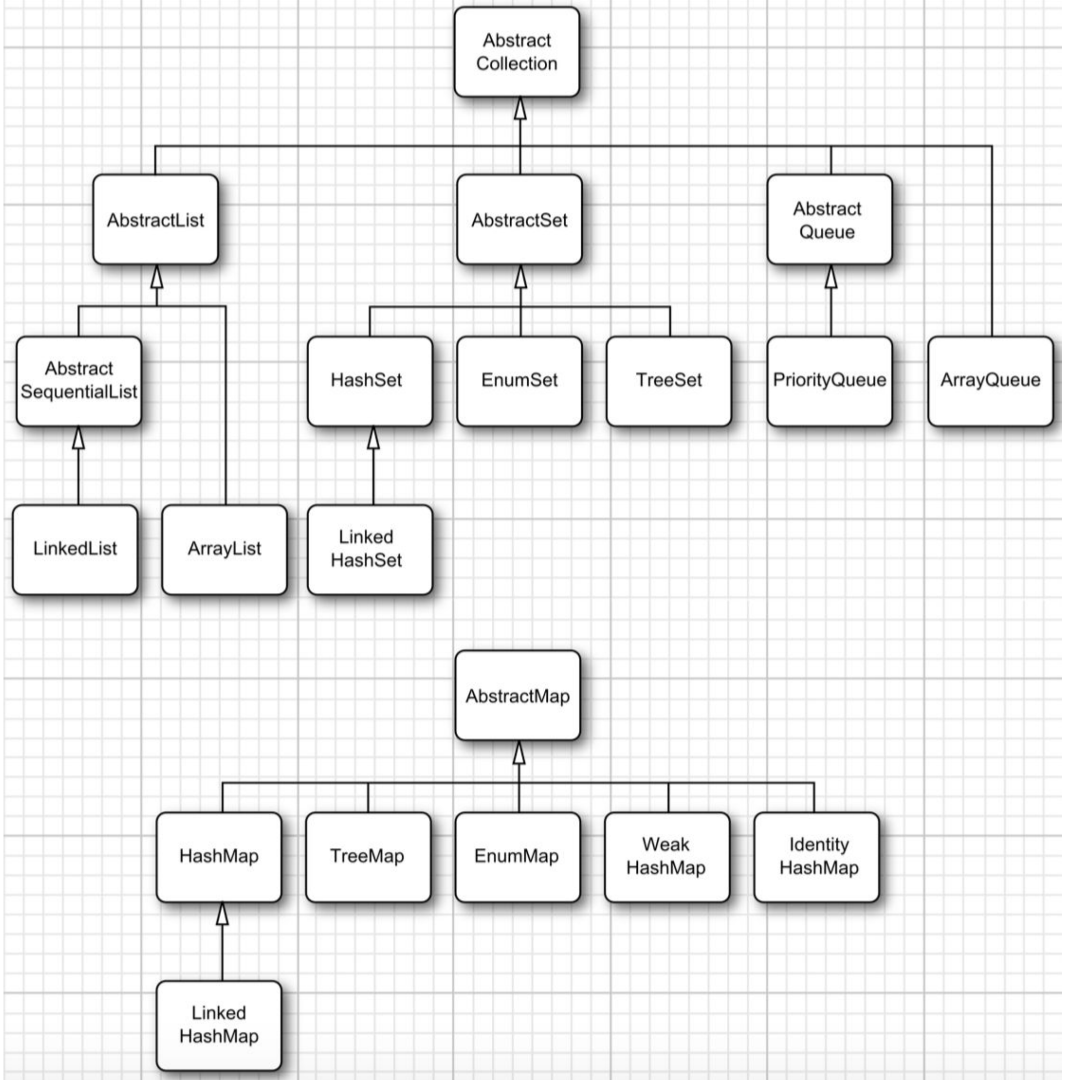
9.1.5 Interfaces in the Collections Framework

- The interfaces of the collections framework that implement the Collection interface
- Two fundamental interfaces for collections: Collection and Map.



9.2 Concrete Collections

- Classes in the collections framework: extend abstract classes.

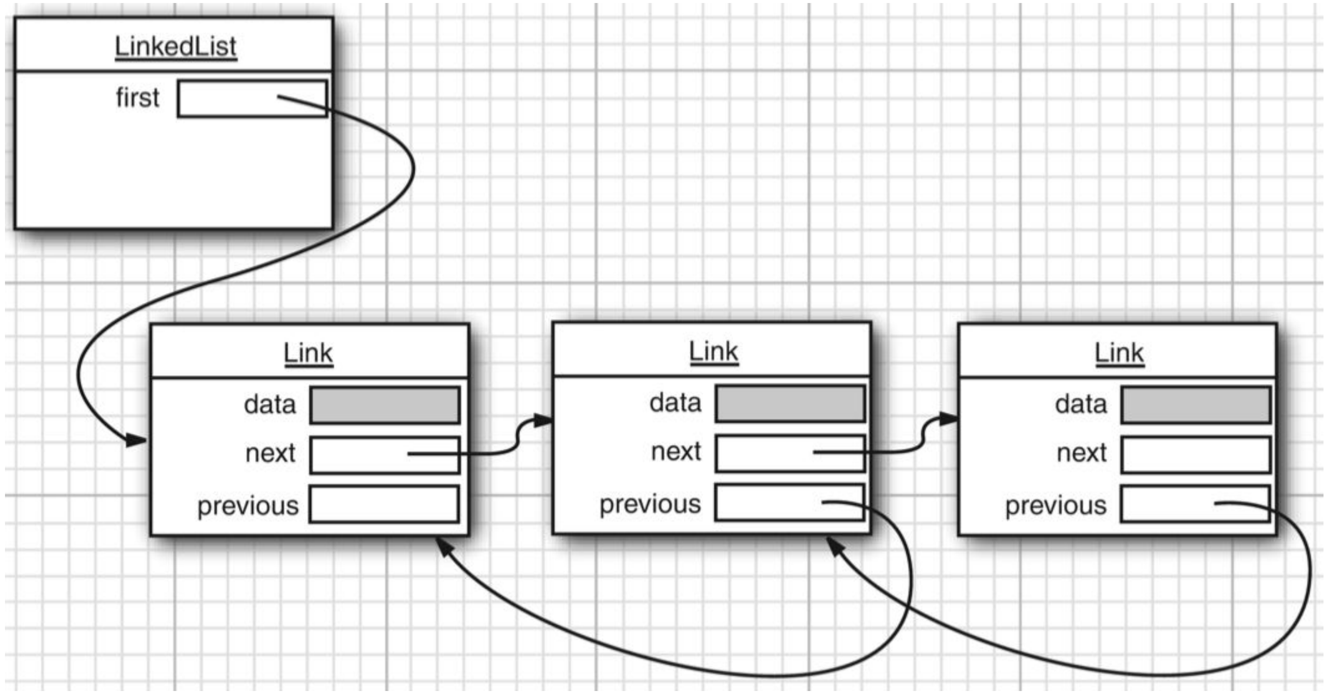


9.2.2 Array Lists

- an ordered collection
- visiting the elements:
    - an iterator
    - random access with methods get and set

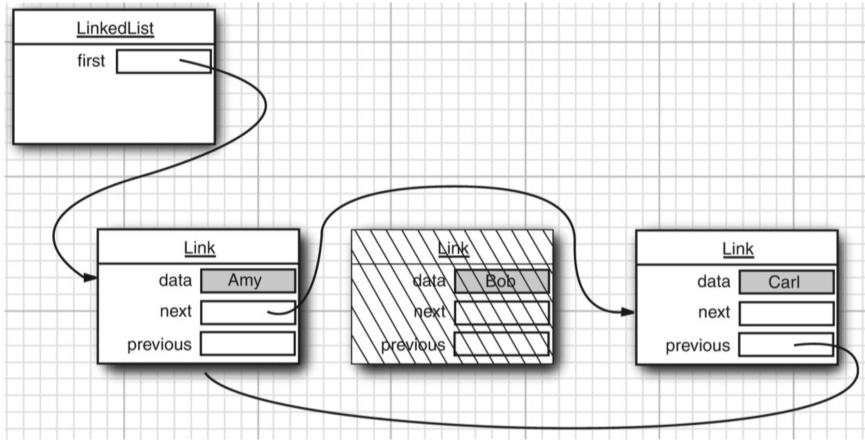
9.2.1 Linked Lists

- Removing/inserting an element from/in the middle of an array: so expensive
- In Java, all linked lists are doubly linked: link to next, previous element.



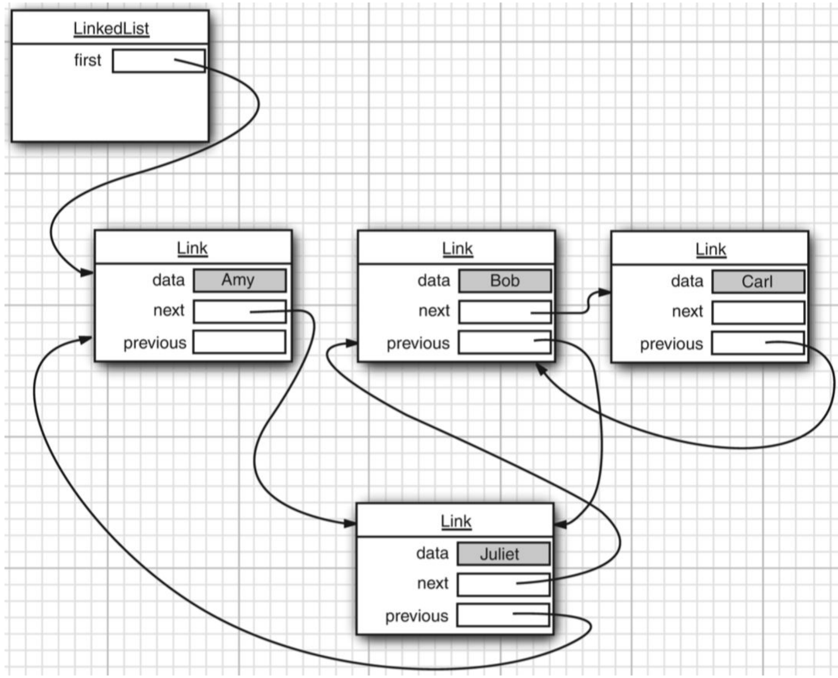
9.2.1 Linked Lists

```
List<String> staff = new LinkedList<>();
// LinkedList implements List
staff.add("Amy");
staff.add("Bob");
staff.add("Carl");

Iterator iter = staff.iterator();
String first = iter.next(); // visit first element String
second = iter.next(); // visit second element
iter.remove(); // remove last visited element
```



9.2.1 Linked Lists

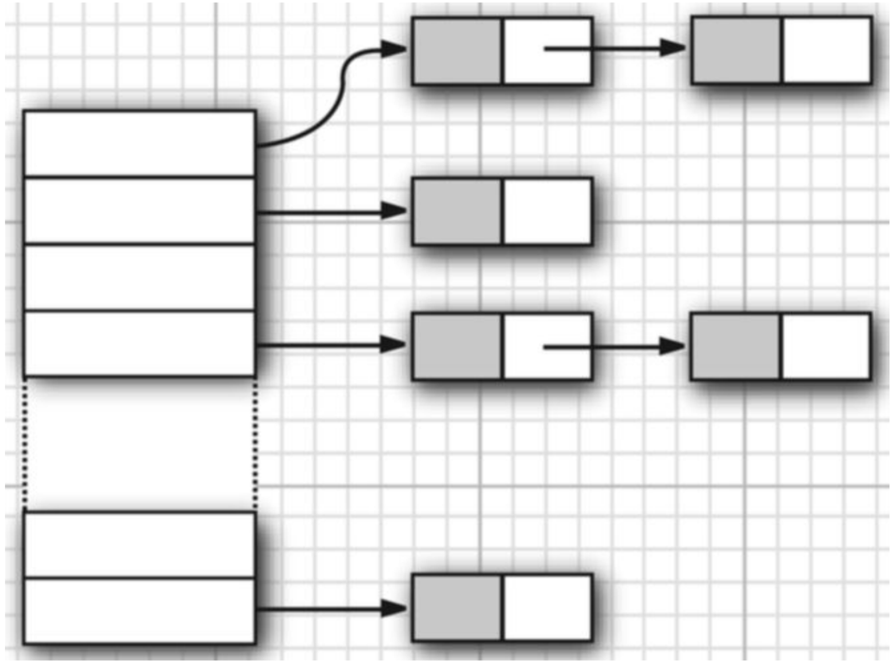
```
List<String> staff = new LinkedList<>();
staff.add("Amy");
staff.add("Bob");
staff.add("Carl");

ListIterator<String> iter =
staff.listIterator();
iter.next(); // skip past first element
iter.add("Juliet");
```



9.2.3 Hash Sets

Hash table helps to find objects quickly
In Java, hash tables are implemented as arrays of linked lists.

- each list is called a *bucket*.
- find the place of an object in the table:
  - compute its hash code
  - reduce it modulo the total number of buckets.



9.2.3 Hash Sets

HashSet class implements a set based on a hash table.

- add elements with **add()**.
- contains(): make a fast lookup to see if an element is already present in the set.
- iterator visits all buckets in turn in a random order.

Full code: set/SetTest.java

9.2.4 Tree Sets

- TreeSet class is similar to the hashset with one added improvement - a sorted collection.
- Every time an element is added to a tree, it is placed into its proper sorting position.
  - The elements must implement the Comparable interface for comparing them.
- Adding an element to a tree is slower than adding it to a hash table.

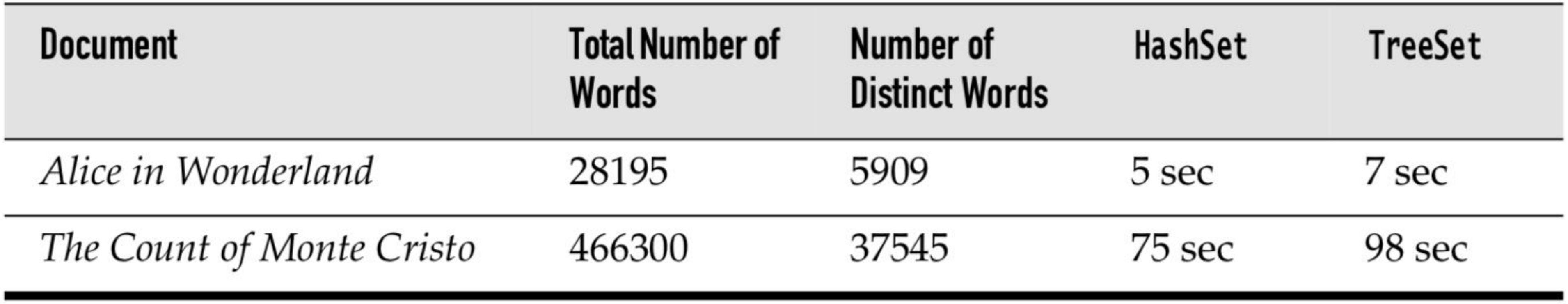


Table 9.3 Adding Elements into Hash and Tree Sets

Document	Total Number of Words	Number of Distinct Words	HashSet	TreeSet
Alice in Wonderland	28195	5909	5 sec	7 sec
The Count of Monte Cristo	466300	37545	75 sec	98 sec

9.2.5 Queues and Deques

- A queue allows efficiently to add elements at the tail and remove elements from the head.
- A double-ended queue, or *deque*, allows efficiently to add or remove elements at the head and tail.
- Adding elements in the middle is not supported.

9.2.6 Priority Queues

- A priority queue retrieves elements in sorted order after they were inserted in arbitrary order.
    - whenever calling the remove method, you get the smallest element currently in the priority queue.
- A priority queue can either hold elements of a class that implements the Comparable interface or a Comparator object
- A typical use for a priority queue is job scheduling.
    - Each job has a priority.
    - Jobs are added in random order.
    - Whenever a new job can be started, the highest priority job is removed from the queue.

9.3 Maps

- A set is a collection that lets you quickly find an existing element. However, you need to have an exact copy of the element to find.
- Usually, you have some key information, and you want to look up the associated element.

The map data structure serves that purpose.

9.3.1 Basic Map Operations

Two general-purpose implementations for maps: HashMap and TreeMap

A hash map hashes the keys whereas a tree map uses an ordering on the keys to organize them in a search tree.

Full code: map/MapTest.java

```
Map<String, Employee> staff = new HashMap<>();
// HashMap implements Map
Employee harry = new Employee("Harry Hacker");
staff.put("987-98-9996", harry);

String id = "987-98-9996";
e = staff.get(id);
// gets harry

staff.forEach((k, v) ->
System.out.println("key=" + k + ", value=" + v));
```

9.3.3 Map Views

- The collections framework does not consider a map itself as a collection.
- Obtain views of the map - objects that implement the Collection interface
- There are three views:
  - the **set** of keys: Set<K> keySet()
  - the collection of values (which is not a **set**): Collection<V> values()
  - the set of key/value pairs: Set<Map.Entry<K, V>> entrySet()

```
Set<String> keys = map.keySet();
for (String key : keys){
     // do something with key }

for (Map.Entry<String, Employee> entry :
staff.entrySet()) {
     String k = entry.getKey();
     Employee v = entry.getValue();
     // do something with k, v
}
```