

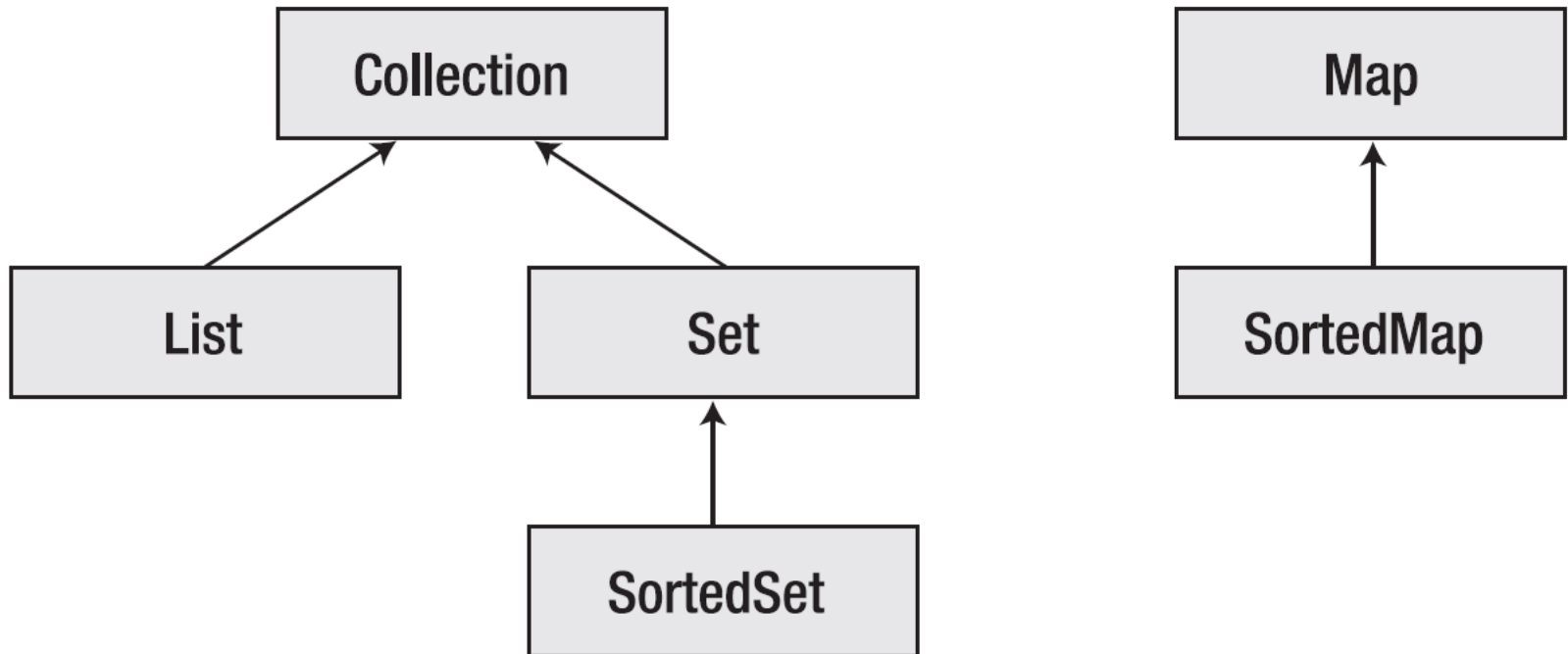
# Java Programming

## Chapter 10 - Collection

# Understanding Collections

- ▶ A collection is an object that organizes multiple data items into a single group.
- ▶ Collections are used to manipulate data, store and retrieve data, and transfer data between methods.
- ▶ ***Collections framework:***
  - Interfaces: abstract data types
  - Implementations: The concrete implementations
  - Algorithms: The methods that perform operations

# The Collections Interfaces



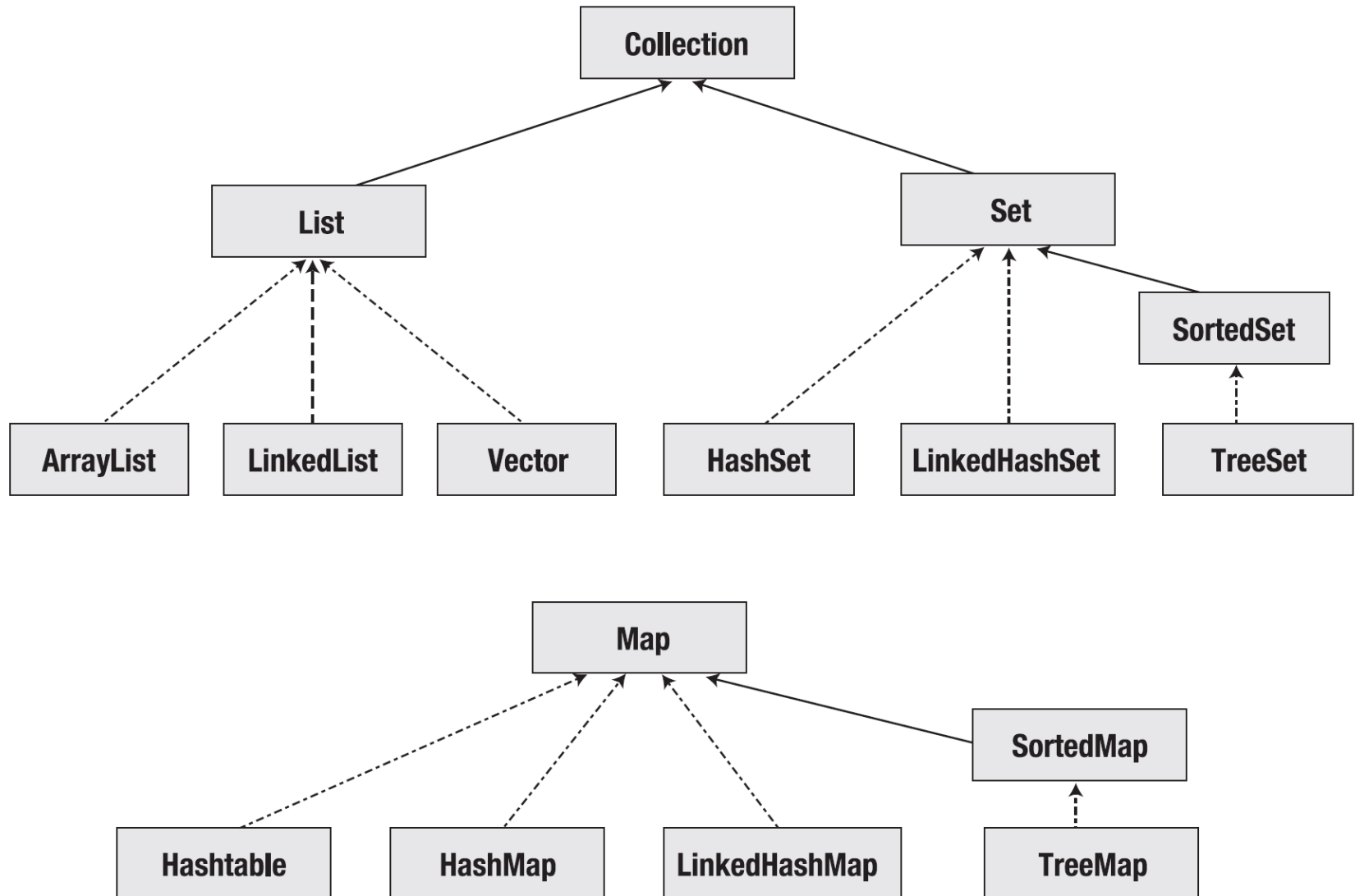
**Table 10-2.** *Some Methods in the Collection Interface*

Method	Description
<code>boolean add(Object obj)</code>	Adds <code>obj</code> (any object reference) to the collection and returns <code>true</code> if an equal object is already not in the collection, else returns <code>false</code> .
<code>boolean addAll(Collection c)</code>	Adds all the elements in the passed-in collection <code>c</code> to this collection. Returns <code>true</code> if the collection changed as a result of the method call.
<code>void clear()</code>	Remove all the elements from this collection.
<code>boolean contains(Object obj)</code>	Returns <code>true</code> if this collection contains the specified element <code>obj</code> .
<code>boolean containsAll(Collection c)</code>	Returns <code>true</code> if this collection contains all the elements of the collection specified in the argument.
<code>boolean isEmpty()</code>	Returns <code>true</code> if this collection has no element in it.
<code>boolean equals(Object obj)</code>	Returns <code>true</code> if the object specified in the argument is equal to this collection.
<code>int hashCode()</code>	Returns the hashCode for this collection as an <code>int</code> type.
<code>boolean remove(Object obj)</code>	Removes the element specified in the argument, and returns <code>true</code> if this collection changes as a result of this method invocation.
<code>boolean removeAll(Collection c)</code>	Removes from this collection all the elements in the collection specified in the method argument, and returns <code>true</code> if this collection changes as a result of this method invocation.
<code>Iterator&lt;E&gt; iterator()</code>	Returns an iterator over the elements (E) in this collection. You will learn more about the angle bracket (<>) notation later in this chapter.
<code>&lt;T&gt; T[] toArray(T[] array)</code>	Returns an array that contains all the elements in this collection. <code>T</code> represents the data type of array elements.

**Table 10-3.** *Methods of the Map Interface*

Method	Description
<code>void clear()</code>	Removes all key-value pairs in the map.
<code>boolean containsKey(Object key)</code>	Returns true if this map contains the key-value pair for the specified key.
<code>boolean containsValue(Object value)</code>	Returns true if this map contains one or more keys corresponding to the specified value.
<code>boolean equals(Object obj)</code>	Returns true if the specified object is equal to this map.
<code>Object get(Object key)</code>	Returns the value corresponding to the specified key.
<code>int hashCode()</code>	Returns the hashCode value for this map.
<code>Boolean isEmpty()</code>	Returns true if this map does not have any key-value pair in it.
<code>Object put(Object key, Object value)</code>	Stores the specified key-value pair. If the key already had a value in the map, it is overwritten with the new value.
<code>void putAll(map m)</code>	Stores all the key-value pairs from the specified map to this map.
<code>Object remove(Object key)</code>	Removes the key-value pair for this key if it is present. Returns the existing value (before the remove operation) corresponding to this key or null if there was no (or null) value.
<code>int size()</code>	Returns the number of key-value pairs in this map.
<code>Collection values()</code>	Returns the values in this map as a Collection.

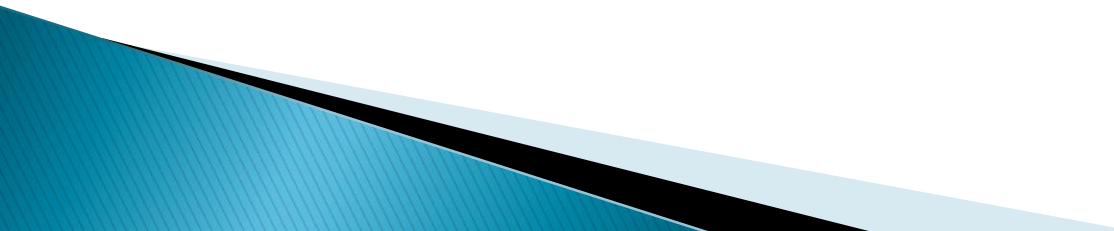
# Implementations of Collections Interfaces



**Table 10-4.** *Some Implementations of Map and Collection Interfaces and Their Characteristics*

Class	Interface	Duplicates Allowed?	Ordered/Sorted	Synchronized
ArrayList	List	Yes	Ordered by index Not sorted	No
LinkedList	List	Yes	Ordered by index Not sorted	No
Vector	List	Yes	Ordered by index Not sorted	Yes
HashSet	Set	No	Not ordered Not sorted	No
LinkedHashSet	Set	No	Ordered by insertion Not sorted	No
TreeSet	Set	No	Sorted either by natural order or by your comparison rules	No
HashMap	Map	No	Not ordered Not sorted	No
LinkedHashMap	Map	No	Ordered	No
Hashtable	Map	No	Not ordered Not sorted	Yes
TreeMap	Map	No	Sorted either by natural order or by your comparison rules	No

# Performing a Search on Collections

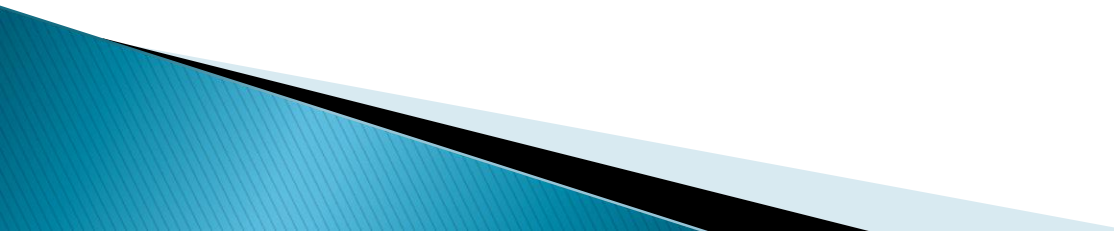
- ▶ The Arrays class and the Collections class present methods that can be used for this purpose.
  - ▶ `binarySearch()`: return the integer index of the searched element if the search is successful.
  - ▶ You must sort the collection (or array) before conducting a search on it.
- 



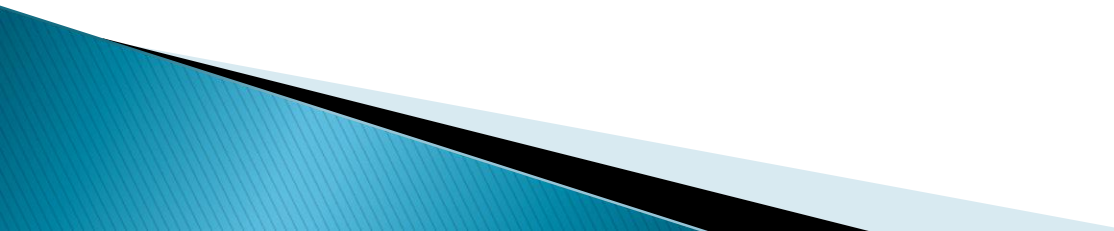
**Listing 10-3. SearchArrayTest.java**

```
1. import java.util.*;
2. class SearchCollectionTest {
3.     public static void main(String[] args) {
4.         String [] str = {"Mark", "Ready", "Set", "Go"};
5.
6.         System.out.println("Unsorted:");
7.         for (String s : str) System.out.print(s + " ");
8.         System.out.println("\nGo = " + Arrays.binarySearch(str, "Go"));
9.
10.        Arrays.sort(str);
11.        System.out.println("Sorted in natural order:");
12.
13.        for (String s : str) System.out.print(s + " ");
14.        System.out.println("\nGo = " + Arrays.binarySearch(str, "Go"));
15.        System.out.println("Sorted in reverse order using a Comparator:");
16.        MyReverseSorter ms = new MyReverseSorter();
17.        Arrays.sort(str, ms);
18.        for (String s : str) System.out.print(s + " ");
19.        System.out.println("\nGo = " + Arrays.binarySearch(str, "Go"));
20.        System.out.println("Go = " + Arrays.binarySearch(str, "Go", ms));
21.    }
22.
23.    class MyReverseSorter implements Comparator<String> {
24.        public int compare(String s1, String s2) {
25.            return s2.compareTo(s1);
26.        }
27.    }
```

# Important Points about Collections

- ▶ Maps have unique keys that facilitate search for their content.
  - ▶ Sets and maps do not allow duplicate entries.
  - ▶ Lists maintain an order, and duplicate elements may exist.
  - ▶ Map implementations do not implement the Collection interface.
- 

# Understanding Generics

- ▶ Reusability of code is an important characteristic of any object-oriented language.
  - ▶ Class inheritance provides reusability of a class.
  - ▶ *Generic programming* enables you to write code that can be reused for different types of objects.
- 

# Generic Collections

- ▶ When you retrieve an element from a collection, you need to cast it to the right type.

1. `ArrayList myList = new ArrayList();`
2. `String st = "Flemingo";`
3. `myList.add(st);`
4. `String st1 = (String) myList.get(0);`
5. `System.out.println(st1);`

- ▶ The code will compile without an error if you replace line 4 with the following line:

```
Integer st1 = (Integer) myList.get(0);
```

1. `ArrayList<String> myList = new ArrayList<String>();`
  2. `String st = "Flemingo";`
  3. `myList.add(st);`
  4. `String st1 = myList.get(0);`
  5. `System.out.println(st1);`
- ▶ You will receive a compiler error if you replace line 4 with the following line:
- ```
Integer st1 = (Integer) myList.get(0);
```
- 