12.2 The Java Collections Framework

- Framework
  - » A collection of interfaces and classes that implement useful data structures and algorithms
- Collections Framework
  - » used to manipulate groups of objects
    - – Bag
    - – Set
    - – Map
    - – List
- HashSet, HashMap

((Abstract Collections))

- 1) Bags
  - an unordered collection of elements , may contain duplicate elements
  - known as multisets
  - least restrictive and most general forms of collections
  - In the Java : Collection interface
  - least restrictive, most general form, rarely used directly
- 2) Sets
  - an unordered collection of elements
  - can not contain duplicate elements
  - a set : $\{e_1, e_2, \ldots, e_n\}$,e2… : elements of the set
  - a set of languages : `{ "English", "Chinese", "German" }`
  - sorted sets, ordered sets : `{ "Chinese", "English", "German" }`
  - In the Java : the Set interface, SortedSet interface

((Abstract Collections))

- 3) Lists
  - » an ordered collection of elements
  - » == sequences
  - » elements are indexed sequentially starting from 0
  - » duplicate elements are allowed
  - » A list : $\langle e_1, e_2, \ldots, e_n \rangle$
  - » Different lists :

```
⟨ "Chinese", "German", "English" ⟩
⟨ "English", "Chinese", "German" ⟩
⟨ "English", "Chinese", "English", "German" ⟩
```

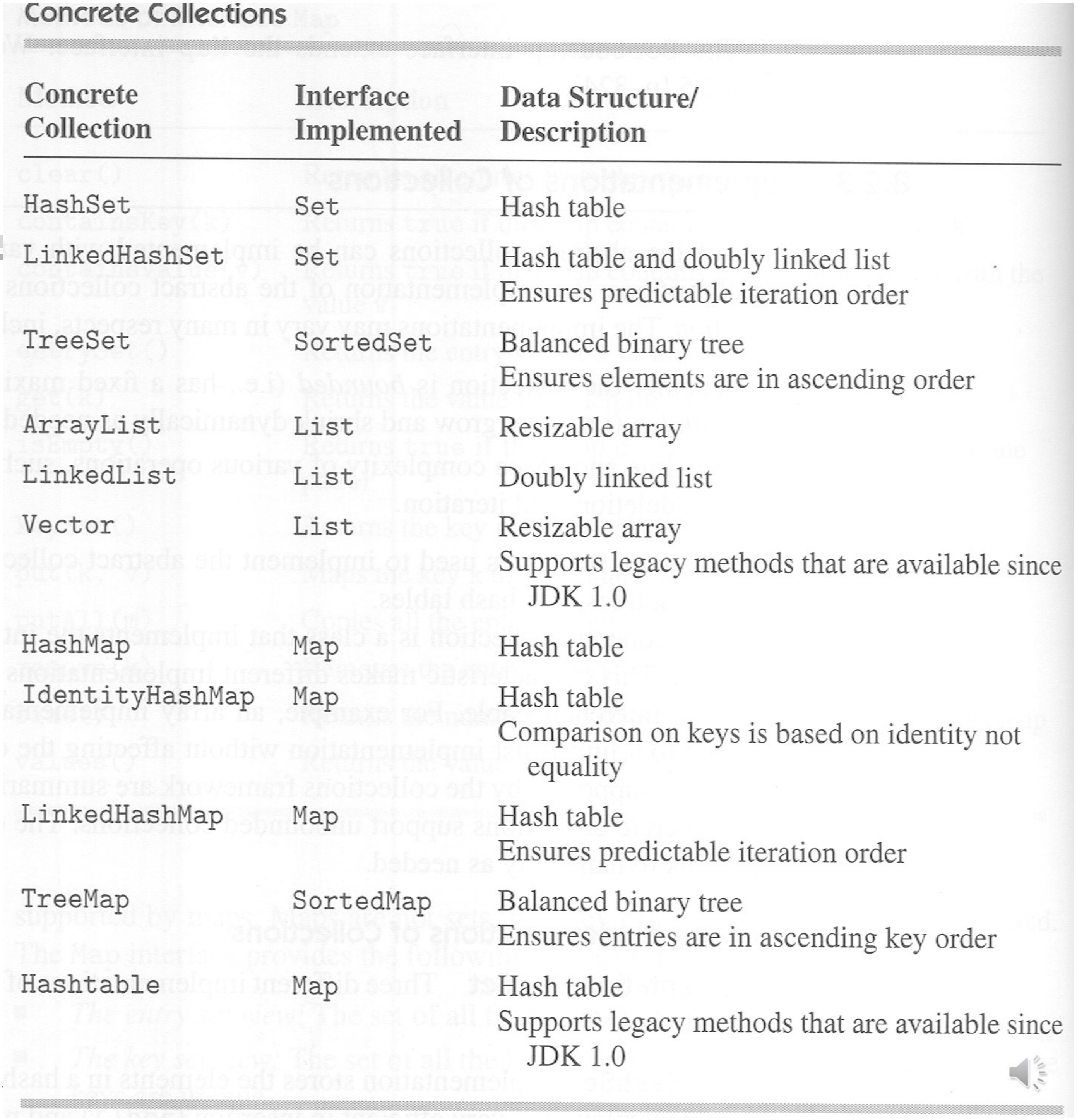
  - » In the Java : List interface

((Abstract Collections))

- 4) Maps
  - » An <u>unordered </u>collection of <u>key-value pairs</u>
  - » Denoted <u>key |→ value</u>
  - » <u>Functions, dictionaries, or associative arrays</u>
  - » The keys in a map must be <u>unique (can map at most one value)</u>
  - » A Map $\{k_1 \mapsto v_1, k_2 \mapsto v_2, \ldots, k_n \mapsto v_n\}$
    - – k1, k2…. : the key. v1, v2…. : the values of the map
  - » Ex) very small English-Chinese dictionary

{ "welcome" ↦ "欢迎", "software" ↦ "软件", "coffee" ↦ "咖啡" }

  - » <u>Sorted maps</u>(== ordered maps) : automatically sorted by keys.
  - » In the Java: <u>Map</u> interface, <u>SortedMap</u> interface



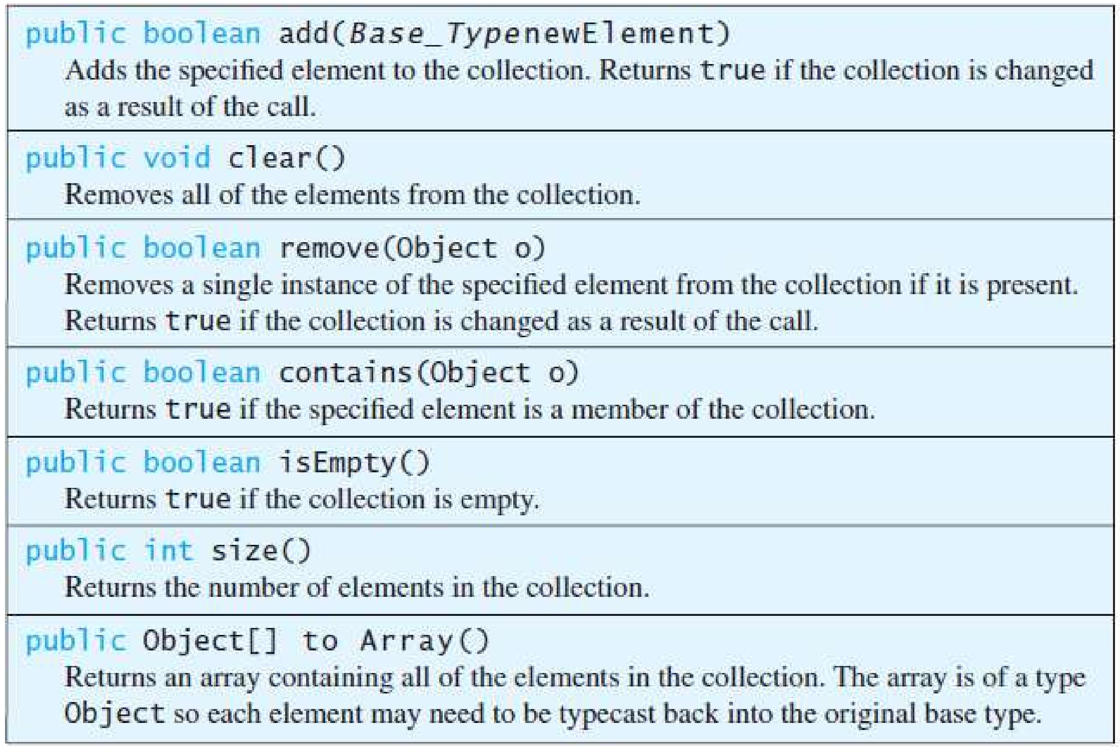
Concrete Collections

Concrete Collection	Interface Implemented	Data Structure/ Description
`HashSet`	`Set`	Hash table
`LinkedHashSet`	`Set`	Hash table and doubly linked list Ensures predictable iteration order
`TreeSet`	`SortedSet`	Balanced binary tree Ensures elements are in ascending order
`ArrayList`	`List`	Resizable array
`LinkedList`	`List`	Doubly linked list
`Vector`	`List`	Resizable array Supports legacy methods that are available since JDK 1.0
`HashMap`	`Map`	Hash table
`IdentityHashMap`	`Map`	Hash table Comparison on keys is based on identity not equality
`LinkedHashMap`	`Map`	Hash table Ensures predictable iteration order
`TreeMap`	`SortedMap`	Balanced binary tree Ensures entries are in ascending key order
`Hashtable`	`Map`	Hash table Supports legacy methods that are available since JDK 1.0

The Collection Interface

- The `Collection` interface specifies how objects can be added, removed, or accessed from a `Collection`

FIGURE 12.2 Selected Methods in the `Collection` Interface



`public boolean add(Base_TypenewElement)`	Adds the specified element to the collection. Returns `true` if the collection is changed as a result of the call.
`public void clear()`	Removes all of the elements from the collection.
`public boolean remove(Object o)`	Removes a single instance of the specified element from the collection if it is present. Returns `true` if the collection is changed as a result of the call.
`public boolean contains(Object o)`	Returns `true` if the specified element is a member of the collection.
`public boolean isEmpty()`	Returns `true` if the collection is empty.
`public int size()`	Returns the number of elements in the collection.
`public Object[] to Array()`	Returns an array containing all of the elements in the collection. The array is of a type `Object` so each element may need to be typecast back into the original base type.

`HashSet` Class

- Used to store a set of objects
- Uses the same `<>` notation as an `ArrayList` to specify the data type
- View source code, listing 12.2
  `class HashSetDemo`
- If you use `HashSet` of your own class, it must override `hashCode()` and `equals()`

LISTING 12.2 A HashSet Demonstration *(part 1 of 2)*

```
import java.util.HashSet;
public class HashSetDemo
{
    public static void main(String[] args)
    {
        HashSet<Integer> intSet = new HashSet<Integer>();
        intSet.add(2);
        intSet.add(7);
        intSet.add(7);
        intSet.add(3);
        printSet(intSet);
        intSet.remove(3);
        printSet(intSet);
        System.out.println("Set contains 2: " +
            intSet.contains(2));
        System.out.println("Set contains 3: " +
            intSet.contains(3));
    }
    public static void printSet(HashSet<Integer> intSet)
    {
        System.out.println("The set contains:");
        for (Object obj : intSet.toArray())
        {
            Integer num = (Integer) obj;
            System.out.println(num.intValue());
        }
    }
}
```

Ignored since 7 is already in the set

Sample Screen Output

```
The set contains:
2
3
7
The set contains:
2
7
Set contains 2: true
Set contains 3: false
```

Class AbstractCollection<E>

java.lang.Object
    java.util.AbstractCollection<E>

`Object[]`	`toArray()` Returns an array containing all of the elements in this collection.
`<T> T[]`	`toArray(T[] a)` Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

The Map Interface

- The Map Interface
  - » deals with Collections of unordered pair
  - » pair as consisting of a key K (to search for) and an associated value V.

`HashMap` Class

- Used like a database to efficiently map from a **key** to an **object**
- Uses the same `<>` notation as an `ArrayList` to specify the data type of both the key and object
- View source code, listing 12.3
  `class HashMapDemo`
  - » If you use `HashMap` of your own class as key, it must override `hashCode()` and `equals()`

FIGURE 12.3 Selected Methods in the `Map` Interface

`public Base_Type_Value put(Base_Type_Key k, Base_Type_Value v)` Associates the value `v` with the key `k`. Returns the previous value for `k` or `null` if there was no previous mapping
`public Base_Type_Value get(Object k)` Returns the value mapped to the key `k` or `null` if no mapping exists.
`public void clear()` Removes a single instance of the specified element from the collection if it is present. Returns `true` if the collection is changed as a result of the call.
`public Base_Type_Value remove(Object k)` Removes the mapping of key `k` from the map if present. Returns the previous value for the key `k` or `null` if there was no previous mapping.
`public boolean containsKey(Object k)` Returns `true` if the key `k` is a key in the map.
`public boolean containsValue(Object v)` Returns `true` if the value `v` is a value in the map.
`public boolean isEmpty()` Returns `true` if the map contains no mappings.
`public int size()` Returns the number of mappings in the map.
`public Set <Base_Type_Key> keySet()` Returns a set containing all of the keys in the map.
`public Collection <Base_Type_Values> values()` Returns a collection containing all of the values in the map.

LISTING 12.3 A HashMap Demonstration *(part 1 of 2)*

```java
import java.util.HashMap;
public class HashMapDemo
{
    public static void main(String[] args)
    {
        HashMap<String, Integer> mountains =
            new HashMap<String, Integer>();
        mountains.put("Everest", 29029);
        mountains.put("K2", 28251);
        mountains.put("Kangchenjunga", 28169);
        mountains.put("Denali", 20335);
        printMap(mountains);
        System.out.println("Denali in the map: " +
            mountains.containsKey("Denali"));
        System.out.println();
        System.out.println("Changing height of Denali.");
        mountains.put("Denali", 20320);
        printMap(mountains);

        System.out.println("Removing Kangchenjunga.");
        mountains.remove("Kangchenjunga");
        printMap(mountains);
    }

    public static void printMap(HashMap<String, Integer> map)
    {
        System.out.println("Map contains:");
        for (String keyMountainName : map.keySet())
        {
            Integer height = map.get(keyMountainName);
            System.out.println(keyMountainName + " --> " +
                height.intValue() + " feet.");
        }
        System.out.println();
    }
}
```

Overwrites the old value for Denali

Sample Screen Output

```
Map contains:
K2 --> 28251 feet.
Denali --> 20355 feet.
Kangchenjunga --> 28169 feet.
Everest --> 29029 feet.
Denali in the map: true
Changing height of Denali.
Map contains:
K2 --> 28251 feet.
Denali --> 20320 feet.
Kangchenjunga --> 28169 feet.
Everest --> 29029 feet.
Removing Kangchenjunga.
Map contains:
K2 --> 28251 feet.
Denali --> 20320 feet.
Everest --> 29029 feet.
```

끝