e-Lite

Java collections framework

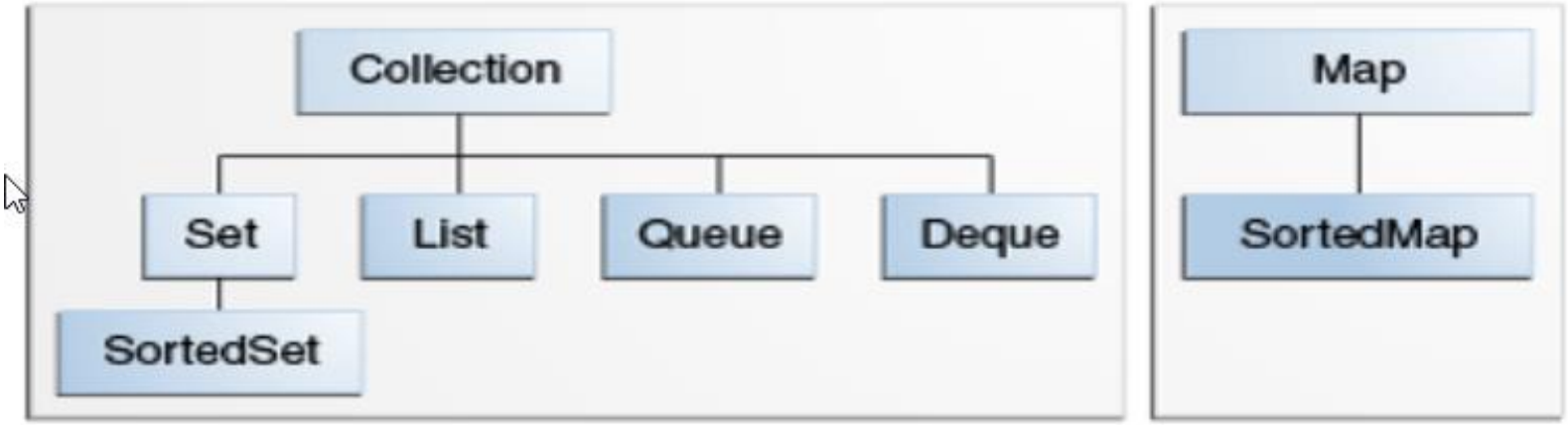
Commonly reusable collection data structures

Java Collections Framework (JCF)

- Collection
  - an object that represents a group of objects
- Collection Framework
  - A unified *architecture* for representing and manipulating collections
  - Such collections are manipulated independent of the details of their representation

Infrastructure

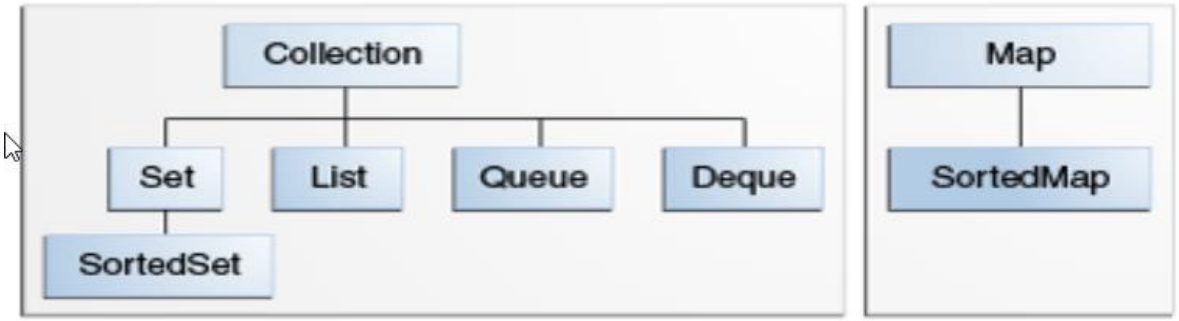
- These interfaces form the basis of the framework
  - Some types of collections **allow duplicate** elements, others do not
  - Some types of collections are **ordered**, others are **unordered**
- The Java platform doesn't provide any direct implementations of the Collection interface, but provides implementations of more specific sub-interfaces, such as Set and List and Maps



https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html

Collection interface

- A **Collection** represents a group of objects known as its *elements*
- The Collection interface is the **least common denominator** that all collections implement.
- It is Used
  - to pass collections around
  - to manipulate them with generally available methods
- **Collection** extends **Iterable**



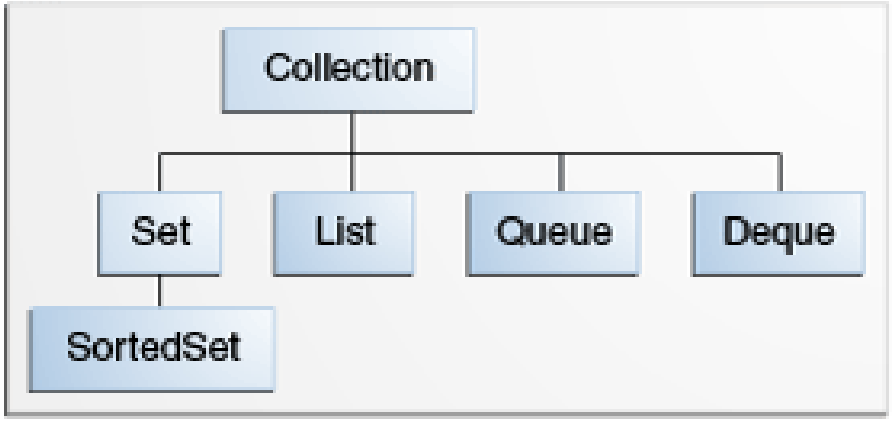
A note on iterators

- An **Iterator** is an object that enables you to traverse through a collection (and to remove elements from the collection selectively)
- You get an Iterator for a collection by calling its iterator() method.
- Several languages supports "iterators". E.g., C++, PHP, Python, Ruby, Go…

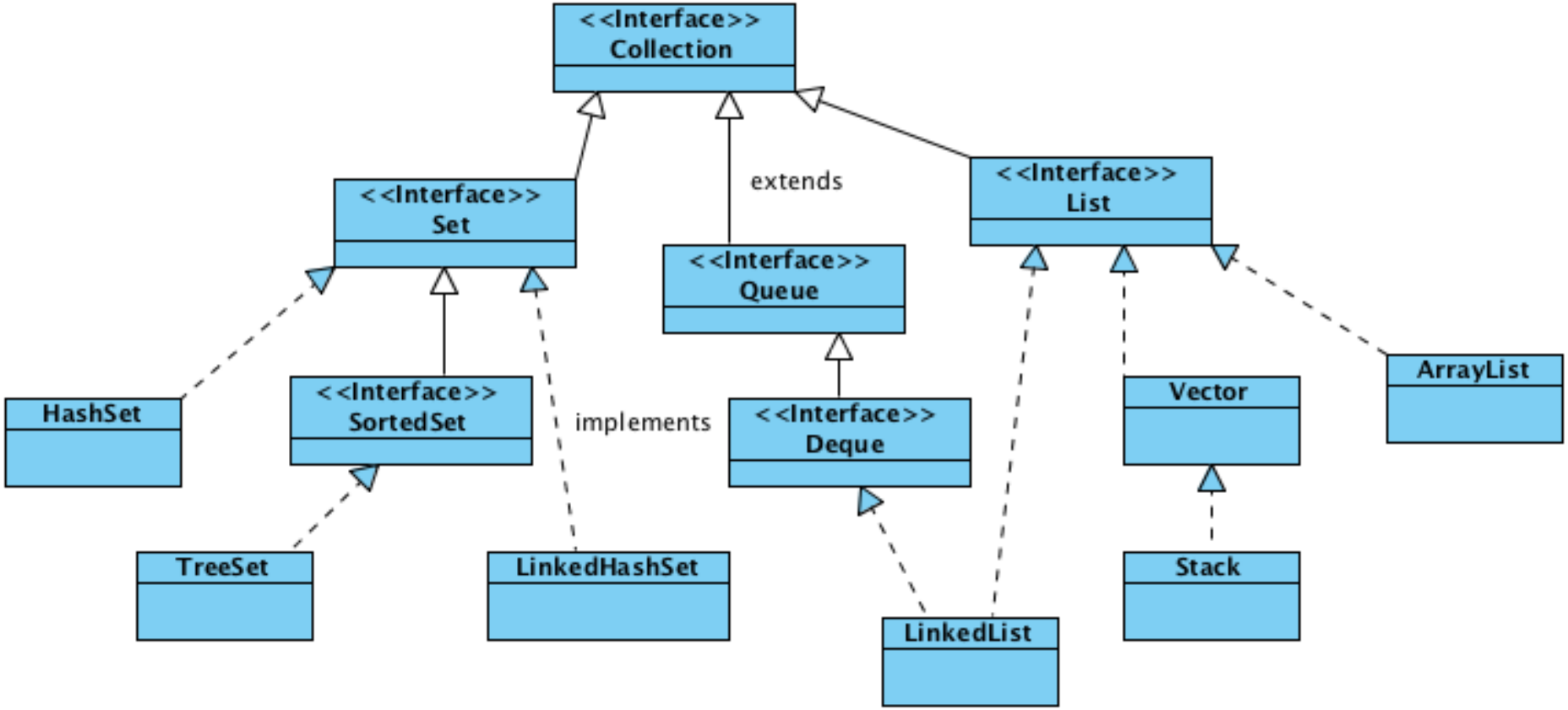
```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

Main Interfaces

- List
  - A more flexible version of an array
- Queue & Priority Queue
  - The order of arrival does matter, or the urgency
- Deque
  - Double-ended Queue (bi-directional)
- Set
  - No order, no duplicate elements



Collection Family Tree



Collection interface

```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);                          //optional
    boolean remove(Object element);                  //optional
    Iterator<E> iterator();

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);              //optional
    boolean retainAll(Collection<?> c);              //optional
    void clear();                                    //optional

    Object[] toArray();
    <T>T[] toArray(T[] a);
}
```

Collection interface

Basic Operations

```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);                          //optional
    boolean remove(Object element);                  //optional
    Iterator<E> iterator();

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);              //optional
    boolean retainAll(Collection<?> c);              //optional
    void clear();                                    //optional

    Object[] toArray();
    <T>T[] toArray(T[] a);
}
```

generics

Collection interface

```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);                          //optional
    boolean remove(Object element);                  //optional

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);        //optional
    boolean retainAll(Collection<?> c);        //optional
    void clear();                              //optional

    Object[] toArray();
    <T>T[] toArray(T[] a);
}
```

Bulk Operations

wildcard

either extends
or implements

Collection interface

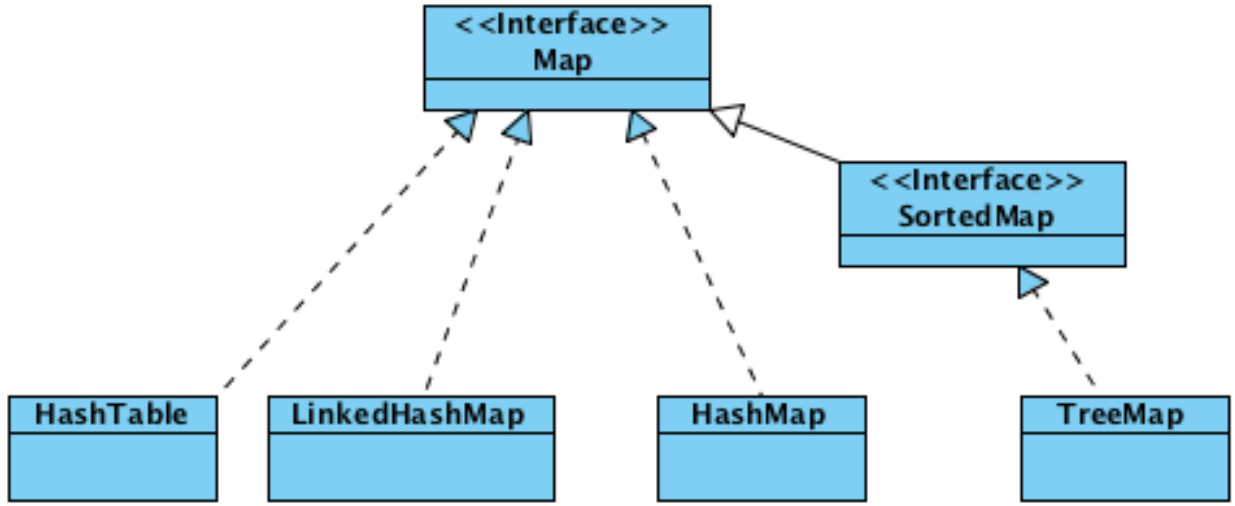
```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);                          //optional
    boolean remove(Object element);                  //optional
    Iterator<E> iterator();

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);              //optional
    boolean retainAll(Collection<?> c);              //optional
                                                     //optional

    Object[] toArray();
    <T>T[] toArray(T[] a);
}
```

Array Operations

Map interface

- A **Map** is an object that maps keys to values
- A map cannot contain duplicate keys: each key can map to at most one value
- **Map** does not extend **Iterable**, but it is possible to get an iterator through **entrySet()**
- Note: Maps do not extend from **java.util.Collection**, but they're still considered to be part of the "collections framework"

Map Family Tree



Map interface

Basic Operations

```
public interface Map<K,V> {
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();

    void putAll(Map<? extends K, ? extends V> m);
    void clear();

    […]
```

Map interface

```
public interface Map<K,V> {
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();

    void putAll(Map<? extends K, ? extends V> m);
    void clear();

    […]
```

Bulk Operations

Map interface

```
    […]

    public Set<K> keySet();
    public Collection<V> values();
    public Set<Entry<K,V>> entrySet();

    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```

Interface for entrySet elements

Map interface

Collection Views

```
public Set<K> keySet();
public Collection<V> values();
public Set<Map.Entry<K,V>> entrySet();

public interface Entry {
    K getKey();
    V getValue();
    V setValue(V
}
}
```

```
for (Map.Entry<Foo,Bar> e : map.entrySet())
{
    Foo key = e.getKey();
    Bar value = e.getValue();
}
```

Implementations

Interfaces	Hash table Implementations	Resizable array Implementations	Tree Implementations	Linked list Implementations	Hash table + Linked list Implementations
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

References

- https://docs.oracle.com/javase/tutorial/collections/

Licenza d'uso

- Queste diapositive sono distribuite con licenza Creative Commons "Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)"
- Sei libero:
  - di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
  - di modificare quest'opera

- Alle seguenti condizioni:
  - **Attribuzione** — Devi attribuire la paternità dell'opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo i cui tu usi l'opera.

  - **Non commerciale** — Non puoi usare quest'opera per fini commerciali.

  - **Condividi allo stesso modo** — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con un licenza identica o equivalente a questa.

- http://creativecommons.org/licenses/by-nc-sa/3.0/