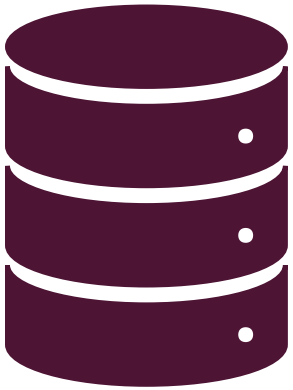


DATA STRUCTURES

WENYE LI
CUHK-SZ

OUTLINE

- Java Collection Framework
- Iterator Interface
- Iterable Interface
- Collection Interface
- List Interface
- Queue Interface / Deque Interface
- Set Interface / SortedSet Interface

This chapter is designed for improving practical skills, not for exam.

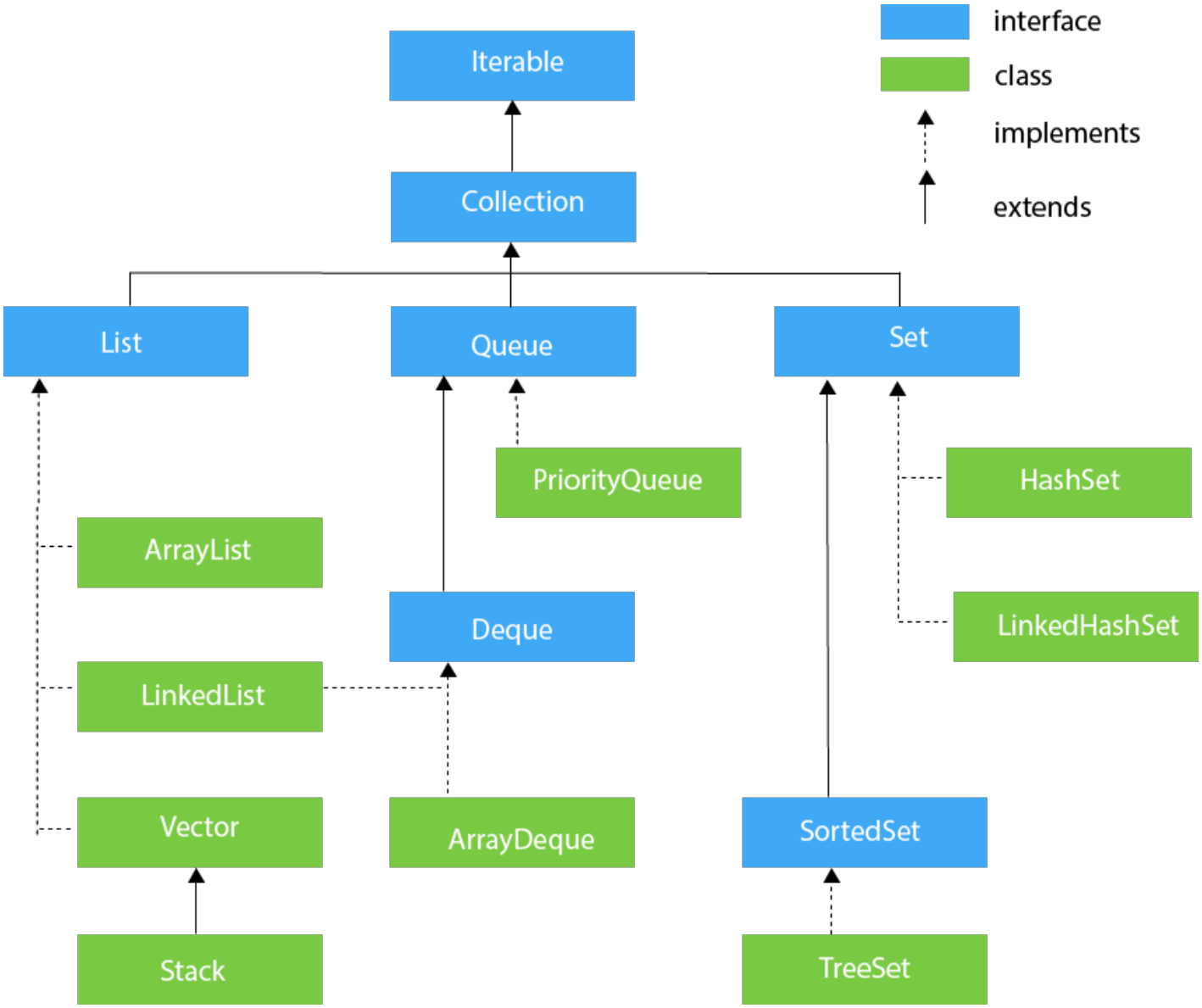
JAVA COLLECTION FRAMEWORK

- What is a framework in Java
  - It provides readymade architecture.
  - It represents a set of classes and interfaces.
  - It is optional.

- What is Collection framework?
  - Represents a unified architecture for storing and manipulating a group of objects.

- Java Collection framework has:
  - Interfaces and its implementations, i.e., classes
  - Algorithm

Hierarchy of Collection Framework



No.	Method	Description
1	public boolean add(E e)	It is used to insert an element in this collection.
2	public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.
4	public boolean removeAll(Collection<?> c)	It is used to delete all the elements of the specified collection from the invoking collection.
5	default boolean removeIf(Predicate<? super E> filter)	It is used to delete all the elements of the collection that satisfy the specified predicate.
6	public boolean retainAll(Collection<?> c)	It is used to delete all the elements of invoking collection except the specified collection.
7	public int size()	It returns the total number of elements in the collection.
8	public void clear()	It removes the total number of elements from the collection.
9	public boolean contains(Object element)	It is used to search an element.
10	public boolean containsAll(Collection<?> c)	It is used to search the specified collection in the collection.
11	public Iterator iterator()	It returns an iterator.
12	public Object[] toArray()	It converts collection into array.
13	public <T> T[] toArray(T[] a)	It converts collection into array. Here, the runtime type of the returned array is that of the specified array.
14	public boolean isEmpty()	It checks if collection is empty.
15	default Stream<E> parallelStream()	It returns a possibly parallel Stream with the collection as its source.
16	default Stream<E> stream()	It returns a sequential Stream with the collection as its source.
17	default Spliterator<E> spliterator()	It generates a Spliterator over the specified elements in the collection.
18	public boolean equals(Object element)	It matches two collections.
19	public int hashCode()	It returns the hash code number of the collection.

ITERATOR INTERFACE

- Iterator interface provides the facility of iterating the elements in a forward direction only.

No.	Method	Description
1	public boolean hasNext()	It returns true if the iterator has more elements otherwise it returns false.
2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is less used.

ITERABLE INTERFACE

- The Iterable interface is the root interface for all the collection classes.
- The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

```
Iterator<T> iterator()
```

It returns the iterator over the elements of type T.

COLLECTION INTERFACE

- The Collection interface is the interface which is implemented by all the classes in the collection framework.

- The Collection interface declares the methods that every collection will have.
  - The Collection interface builds the foundation of the framework.

- Some of the methods:
  - Boolean *add (Object obj)*, Boolean *addAll (Collection c)*, void *clear()*, etc.
  - implemented by all the subclasses of Collection interface.

LIST INTERFACE

- List interface is the child interface of Collection interface.
- It inhibits a list type data structure which can store the ordered collection of objects.
- It can have duplicate values.
- List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

```
List <data-type> list1= new ArrayList();
List <data-type> list2 = new LinkedList();
List <data-type> list3 = new Vector();
List <data-type> list4 = new Stack();
```

There are various methods in List interface that can be used to insert, delete, and access the elements from the list.

```
class TestJavaCollection1
{
    public static void main(String args[])
    {
        ArrayList<String> list = new ArrayList<String>(); //Creating arraylist
        list.add("Ravi");//Adding object in arraylist
        list.add("Vijay");
        list.add("Ravi");
        list.add("Ajay");
        //Traversing list through Iterator
        Iterator itr = list.iterator();
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

```
Ravi
Vijay
Ravi
Ajay
```

The ArrayList class implements the List interface.
It uses a dynamic array to store the duplicate element of different data types.
The ArrayList class maintains the insertion order and is non-synchronized.
The elements stored in the ArrayList class can be randomly accessed.

```java
import java.util.*;
public class TestJavaCollection2
{
    public static void main(String args[])
    {
        LinkedList<String> al = new LinkedList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");
        Iterator<String> itr = al.iterator();
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

```
Ravi
Vijay
Ravi
Ajay
```

LinkedList implements the Collection interface.
It uses a doubly linked list internally to store the elements. It can store the duplicate elements.
It maintains the insertion order and is non-synchronized.
In LinkedList, the manipulation is fast because no shifting is required.

```java
import java.util.*;
public class TestJavaCollection3
{
    public static void main(String args[])
    {
        Vector<String> v = new Vector<String>();
        v.add("Ayush");
        v.add("Amit");
        v.add("Ashish");
        v.add("Garima");
        Iterator<String> itr = v.iterator();
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

```
Ayush
Amit
Ashish
Garima
```

Vector uses a dynamic array to store the data elements.
It is similar to ArrayList.
However, It is synchronized and contains many methods that are not the part of Collection framework.

```java
import java.util.*;
public class TestJavaCollection4
{
    public static void main(String args[])
    {
        Stack<String> stack = new Stack<String>();
        stack.push("Ayush");
        stack.push("Garvit");
        stack.push("Amit");
        stack.push("Ashish");
        stack.push("Garima");
        stack.pop();
        Iterator<String> itr = stack.iterator();
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

```
Ayush
Garvit
Amit
Ashish
```

Stack is the subclass of Vector.
It implements the last-in-first-out data structure, i.e., Stack.
The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

QUEUE INTERFACE

- Queue interface maintains the first-in-first-out order.
- It can be defined as an ordered list that is used to hold the elements to be processed.
- Classes PriorityQueue, Deque, and ArrayDeque implement the Queue interface.

Queue interface can be instantiated as:

```
Queue<String> q1 = new PriorityQueue();
Queue<String> q2 = new ArrayDeque();
```

```java
import java.util.*;
public class TestJavaCollection5
{
    public static void main(String args[])
    {
        PriorityQueue<String> queue = new PriorityQueue<String>();
        queue.add("Amit Sharma");
        queue.add("Vijay Raj");
        queue.add("JaiShankar");
        queue.add("Raj");
        System.out.println("head:" + queue.element());
        System.out.println("head:" + queue.peek());
        System.out.println("iterating the queue elements:");
        Iterator itr = queue.iterator();
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
        queue.remove();
        queue.poll();
        System.out.println("after removing two elements:");
        Iterator<String> itr2 = queue.iterator();
        while(itr2.hasNext())
        {
            System.out.println(itr2.next());
        }
    }
}
```

```
head:Amit Sharma
head:Amit Sharma
iterating the queue elements:
Amit Sharma
Raj
JaiShankar
Vijay Raj
after removing two elements:
Raj
Vijay Raj
```

PriorityQueue implements the Queue interface.
It holds the elements or objects which are to be processed by their priorities.
PriorityQueue doesn't allow null values to be stored in the queue.

DEQUE INTERFACE

- Deque interface extends the Queue interface.
- A double-ended queue which enables us to perform the operations at both the ends.

Deque can be instantiated as:

```
Deque d = new ArrayDeque();
```

```java
import java.util.*;
public class TestJavaCollection6
{
    public static void main(String[] args)
    {
        //Creating Deque and adding elements
        Deque<String> deque = new ArrayDeque<String>();
        deque.add("Gautam");
        deque.add("Karan");
        deque.add("Ajay");
        //Traversing elements
        for (String str : deque)
        {
            System.out.println(str);
        }
    }
}
```

```
Gautam
Karan
Ajay
```

ArrayDeque class implements the Deque interface.
Unlike queue, we can add or delete the elements from both ends.
ArrayDeque is faster than ArrayList and Stack, and has no capacity restrictions.

SET INTERFACE

- Set Interface extends the Collection interface.
- It represents the unordered set of elements which doesn't allow duplicate items.
- We can store at most one null value in Set.
- Set is implemented by HashSet, LinkedHashSet, and TreeSet.

Set can be instantiated as:

```
Set<data-type> s1 = new HashSet<data-type>();
Set<data-type> s2 = new LinkedHashSet<data-type>();
Set<data-type> s3 = new TreeSet<data-type>();
```

```java
import java.util.*;
public class TestJavaCollection7
{
    public static void main(String args[])
    {
        //Creating HashSet and adding elements
        HashSet<String> set = new HashSet<String>();
        set.add("Ravi");
        set.add("Vijay");
        set.add("Ravi");
        set.add("Ajay");
        //Traversing elements
        Iterator<String> itr = set.iterator();
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

```
Vijay
Ravi
Ajay
```

HashSet class implements Set Interface.
It represents the collection that uses a hash table for storage.
Hashing is used to store the elements in the HashSet.
It contains unique items.

```java
import java.util.*;
public class TestJavaCollection8
{
    public static void main(String args[])
    {
        LinkedHashSet<String> set = new LinkedHashSet<String>();
        set.add("Ravi");
        set.add("Vijay");
        set.add("Ravi");
        set.add("Ajay");
        Iterator<String> itr = set.iterator();
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

```
Ravi
Vijay
Ajay
```

LinkedHashSet class represents the LinkedList implementation of Set Interface.
It extends the HashSet class and implements Set interface.
Like HashSet, it also contains unique elements.
It maintains the insertion order and permits null elements.

SORTEDSET INTERFACE

- SortedSet is the alternate of Set interface that provides a total ordering on its elements.
- The elements of the SortedSet are arranged in the increasing (ascending) order.
- It provides additional methods that inhibit the natural ordering of the elements.

The SortedSet can be instantiated as:

```
SortedSet<data-type> set = new TreeSet();
```

```java
import java.util.*;
public class TestJavaCollection9
{
    public static void main(String args[])
    {
        //Creating and adding elements
        TreeSet<String> set = new TreeSet<String>();
        set.add("Ravi");
        set.add("Vijay");
        set.add("Ravi");
        set.add("Ajay");
        //traversing elements
        Iterator<String> itr = set.iterator();
        while(itr.hasNext())
        {
            System.out.println(itr.next());
        }
    }
}
```

```
Ajay
Ravi
Vijay
```

TreeSet class implements the Set interface that uses a tree for storage.
Like HashSet, TreeSet also contains unique elements.
The access and retrieval time of TreeSet is quite fast.
The elements in TreeSet stored in ascending order.

QUESTIONS FOR SELF-STUDY

- What are the two ways to iterate the elements of a collection?
- What is the difference between ArrayList and LinkedList classes in collection framework?
- What is the difference between ArrayList and Vector classes in collection framework?
- What is the difference between HashSet and HashMap classes in collection framework?
- What is the difference between HashMap and Hashtable class?
- What is the difference between Iterator and Enumeration interface in collection framework?
- How can we sort the elements of an object? What is the difference between Comparable and Comparator interfaces?
- What does the hashcode() method do?
- What is the difference between Java collection and Java collections?

THANKS