Java - Introduction to Programming
Lecture 13

Java Collection Framework:

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

A Collection represents a single unit of objects, i.e., a group. The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

What is Framework?

A framework provides a ready-made structure of classes and interfaces for building software applications efficiently.

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

What is Collection framework?

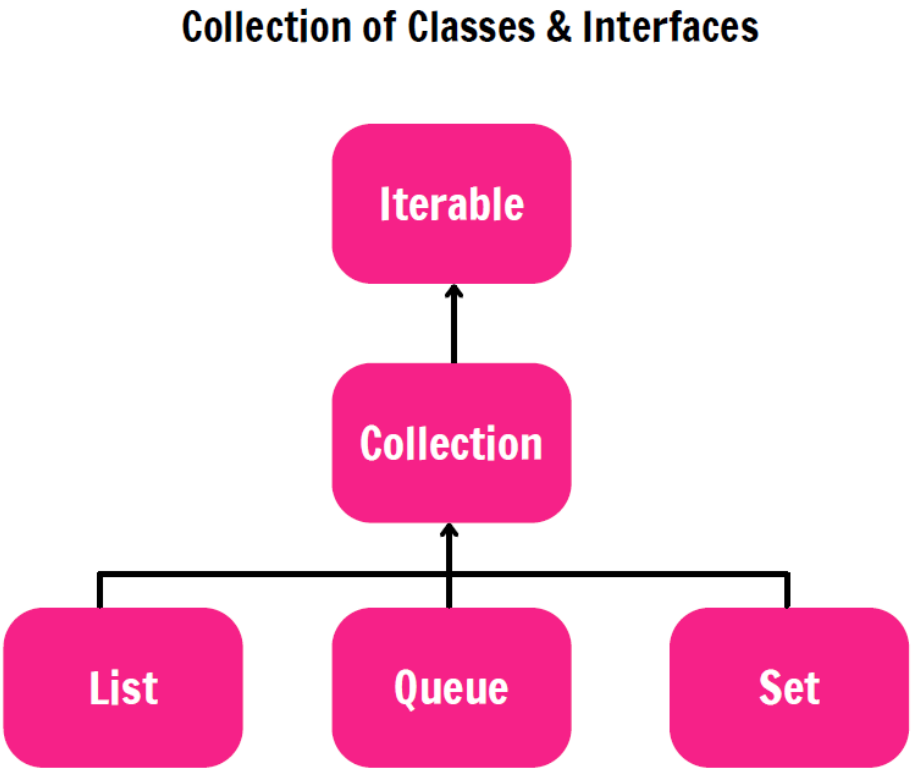
The Collection framework represents a unified architecture for storing and manipulating a group of objects. It enhances code efficiency and readability by offering various data structures, including arrays, linked lists, trees, and hash tables. It has:

- Interfaces and its implementations, i.e., classes
- Algorithm

Hierarchy of Collection Framework

The java.util package contains all the classes and interfaces for the Collection framework. The Java Collections Framework contains following interfaces-
Collection, List, Set, Queue, and Map.

JAVA

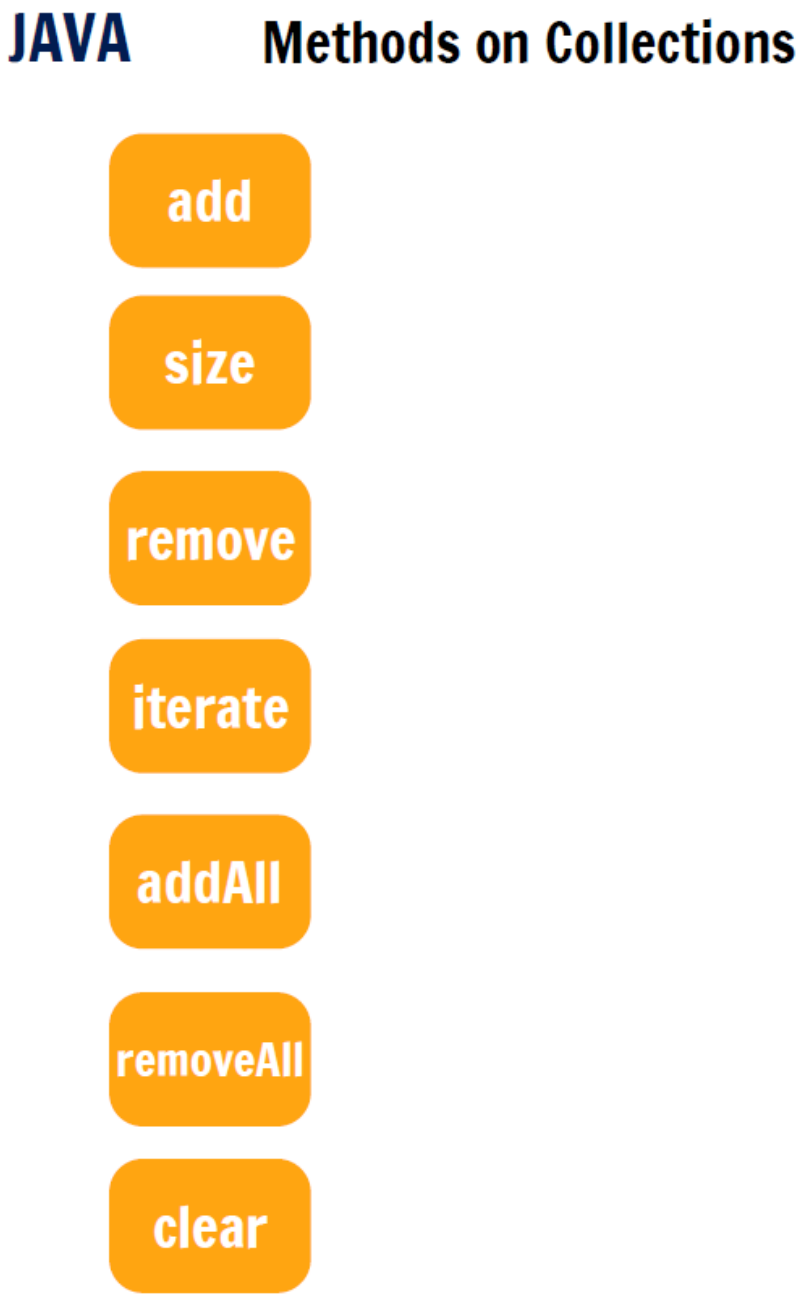


Iterable Interface

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 implement the Iterable interface.

Collection Interface:

The Collection interface is the interface, which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

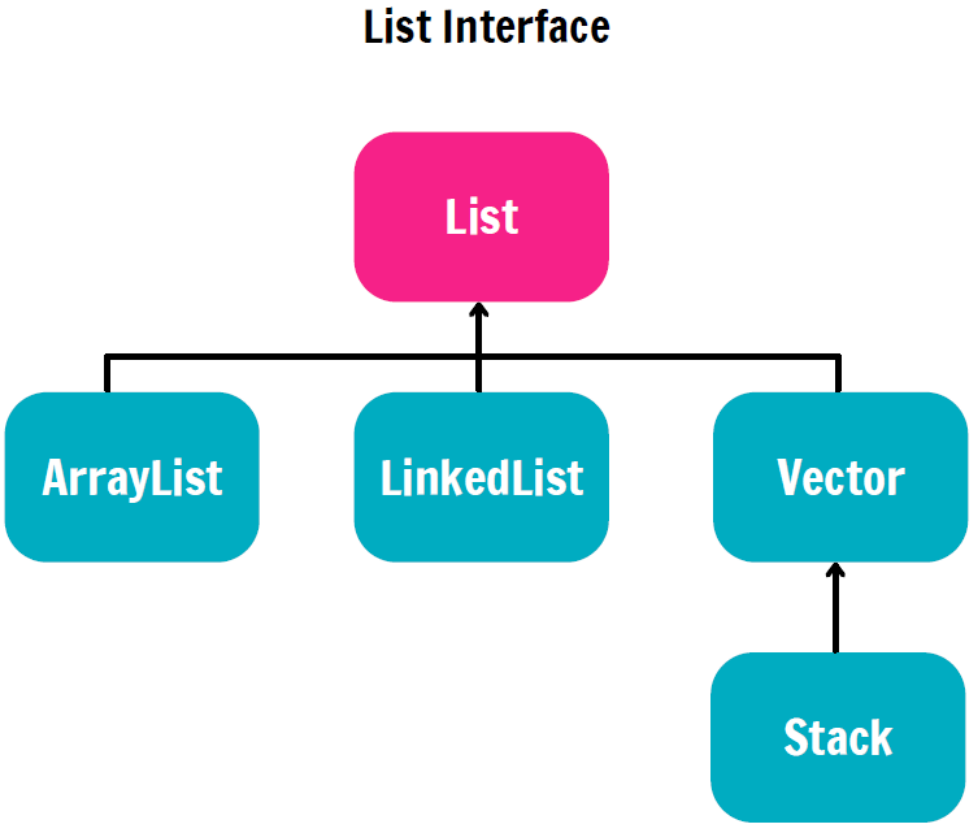


List Interface

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

The classes ArrayList, LinkedList, Vector, and Stack implement list interface.

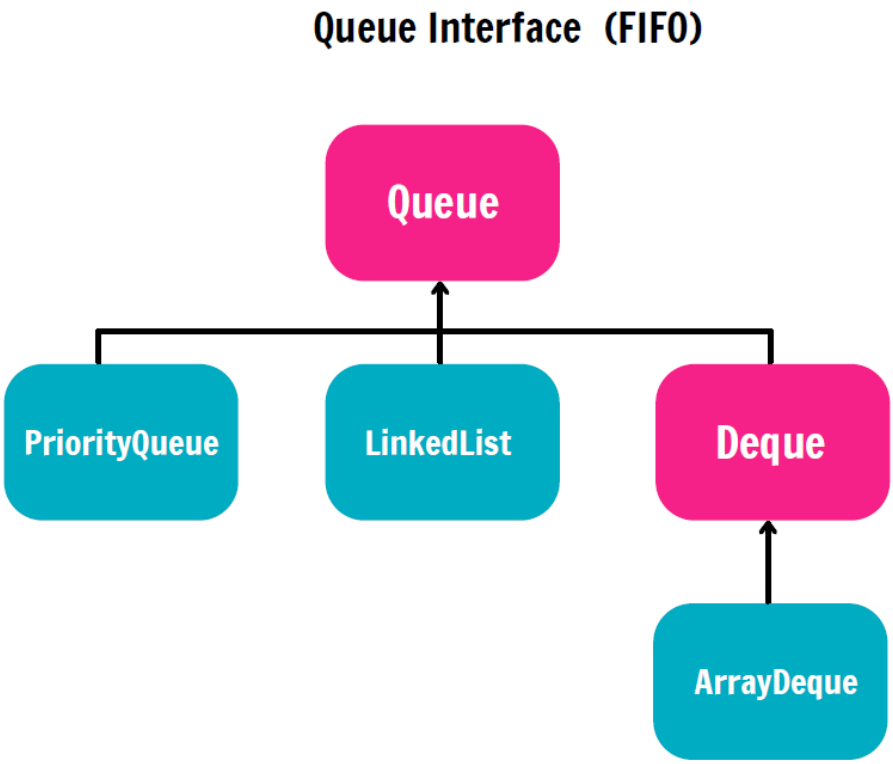
JAVA



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 which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

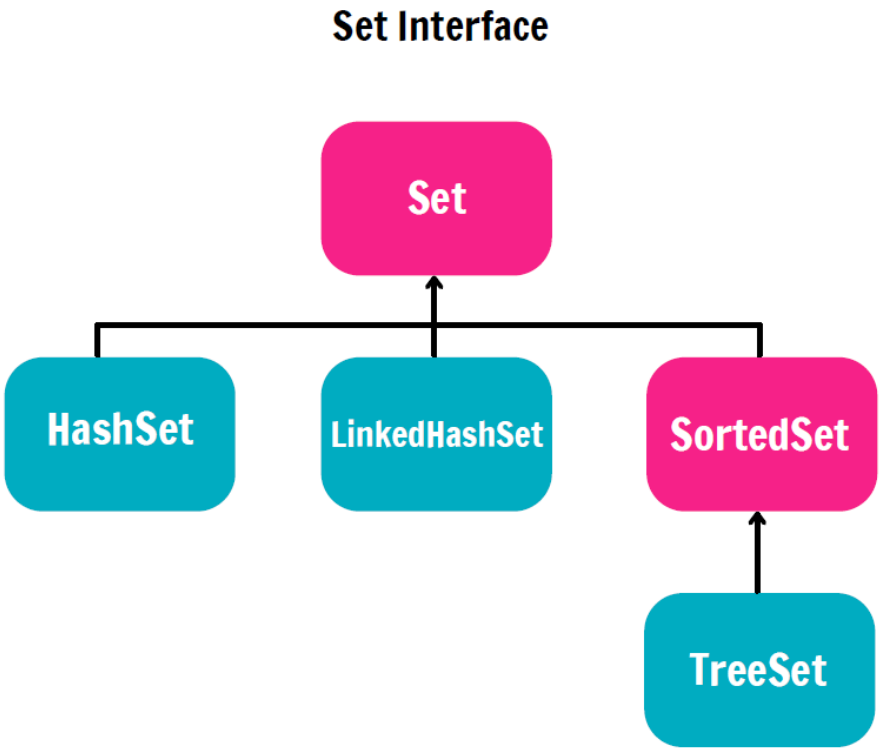
JAVA



Set Interface

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

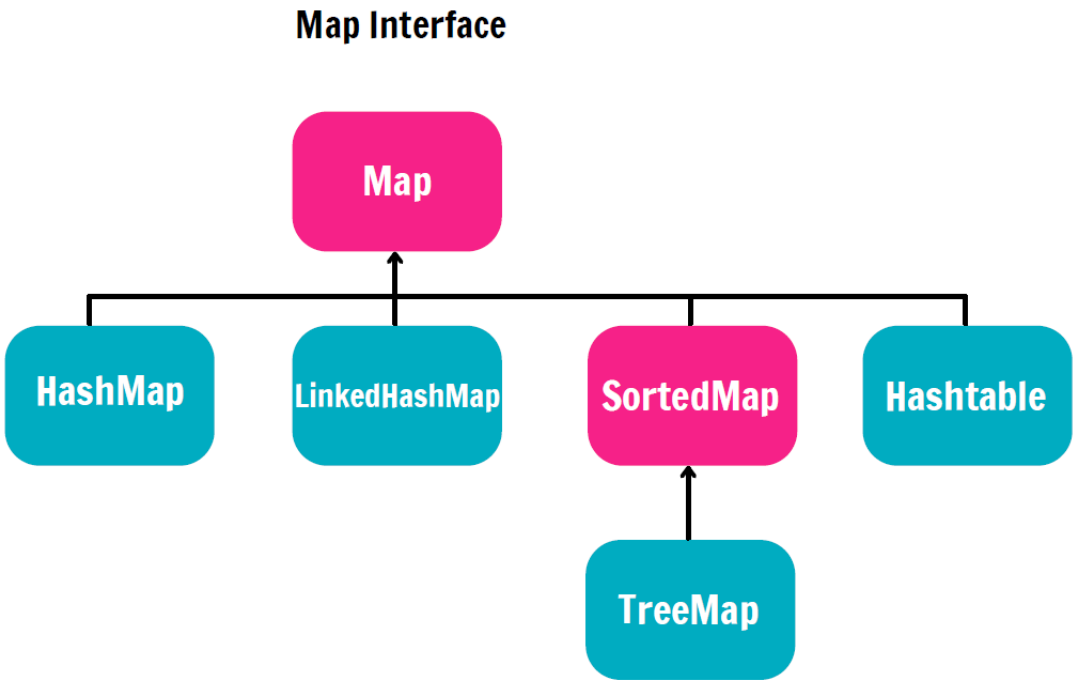
JAVA



Map Interface

The Map interface in Java is part of the Java Collections Framework. It represents a mapping between a set of keys and their corresponding values. A Map cannot contain duplicate keys; each key can map to at most one value. The Map interface is used to store key-value pairs, where each key is unique, and it provides an efficient way to retrieve, update, and manipulate data based on keys.

JAVA



Limitations of Array:

- an Array is a fixed-sized
- Arrays support only elements of the same type
- **Insertion and Deletion Challenges:** Adding or removing elements requires shifting subsequent elements, making these operations inefficient

Array List in Java

ArrayList is a part of collections framework and it is a class of java.util package. It provides us with dynamic sized arrays in Java.

- The main advantage of ArrayList is, unlike normal arrays, we don't need to mention the size when creating ArrayList. It automatically adjusts its capacity as elements are added or removed.
- It may be slower than standard arrays, but helpful when size is not known in advance. Note that creating large fixed sized array would cause wastage of space.
- elements can be added and removed from an ArrayList whenever you want.

Operations:

1. **Declare an ArrayList of different Types**

```
package myPackage;
```

```java
import java.util.ArrayList;

public class ArrayLists {
    public static void main(String[] args) {
        ArrayList<Integer>list1 = new ArrayList<Integer>();
        ArrayList<String>list2 = new ArrayList<String>();
        ArrayList<Float>list3 = new ArrayList<Float>();
        ArrayList<Boolean>list4 = new ArrayList<>();
    }
}
```

2. **Add Element**

```java
package myPackage;
import java.util.ArrayList;

public class ArrayLists {
    public static void main(String[] args) {
        ArrayList<Integer>list = new ArrayList<Integer>();
       //add element
        list.add(1);
        list.add(4);
        list.add(6);
        System.out.println(list);
    }
}
```

3. **Get Element**

```java
package myPackage;
import java.util.ArrayList;

public class ArrayLists {
    public static void main(String[] args) {
        ArrayList<Integer>list = new ArrayList<Integer>();
       //add element
        list.add(1);
        list.add(4);
        list.add(6);
        System.out.println(list);

    //get element
    int element = list.get(0);
    System.out.println(element);
}}
```

4. **Add Element at a specific Index**

```
package myPackage;
import java.util.ArrayList;

public class ArrayLists {
    public static void main(String[] args) {
        ArrayList<Integer>list = new ArrayList<Integer>();
       //add element
        list.add(1);
        list.add(4);
        list.add(6);
        System.out.println(list);

    //add element in between
    list.add(2, 2);
    System.out.println(list);
}}
```

5. **Set Element at a specific Index**

```
package myPackage;
import java.util.ArrayList;

public class ArrayLists {
    public static void main(String[] args) {
        ArrayList<Integer>list = new ArrayList<Integer>();
       //add element
        list.add(1);
        list.add(4);
        list.add(6);
        System.out.println(list);

    //set element
    list.set(0, 5);
    System.out.println(list);
}}
```

6. **Delete Element from an Index**

```
package myPackage;
import java.util.ArrayList;

public class ArrayLists {
    public static void main(String[] args) {
        ArrayList<Integer>list = new ArrayList<Integer>();
       //add element
```

```
        list.add(1);
        list.add(4);
        list.add(6);
        System.out.println(list);
        //delete element
        list.remove(0);
        System.out.println(list);
    }}
```

7. Size of the List

```
package myPackage;
import java.util.ArrayList;

public class ArrayLists {
    public static void main(String[] args) {
        ArrayList<Integer>list = new ArrayList<Integer>();
       //add element
        list.add(1);
        list.add(4);
        list.add(6);
        System.out.println(list);
        //Size of the list
        int size = list.size();
        System.out.println(size);
    }}
```

8. Loop/Iterate on the List

```
package myPackage;
import java.util.ArrayList;

public class ArrayLists {
    public static void main(String[] args) {
        ArrayList<Integer>list = new ArrayList<Integer>();
       //add element
        list.add(1);
        list.add(4);
        list.add(6);
        System.out.println(list);

        //loops
        for(int i=0;i<list.size();i++){
            System.out.println(list.get(i));
        }
    }}
```

9. **Sort the List**

```
package myPackage;
import java.util.ArrayList;
import java.util.Collections;

public class ArrayLists {
    public static void main(String[] args) {
        ArrayList<Integer>list = new ArrayList<Integer>();
       //add element
        list.add(15);
        list.add(14);
        list.add(6);
        System.out.println(list);

        //sorting
        Collections.sort(list);
        System.out.println(list);
        }
    }
```

Homework Problems

Try solving all problems of arrays with arraylists.