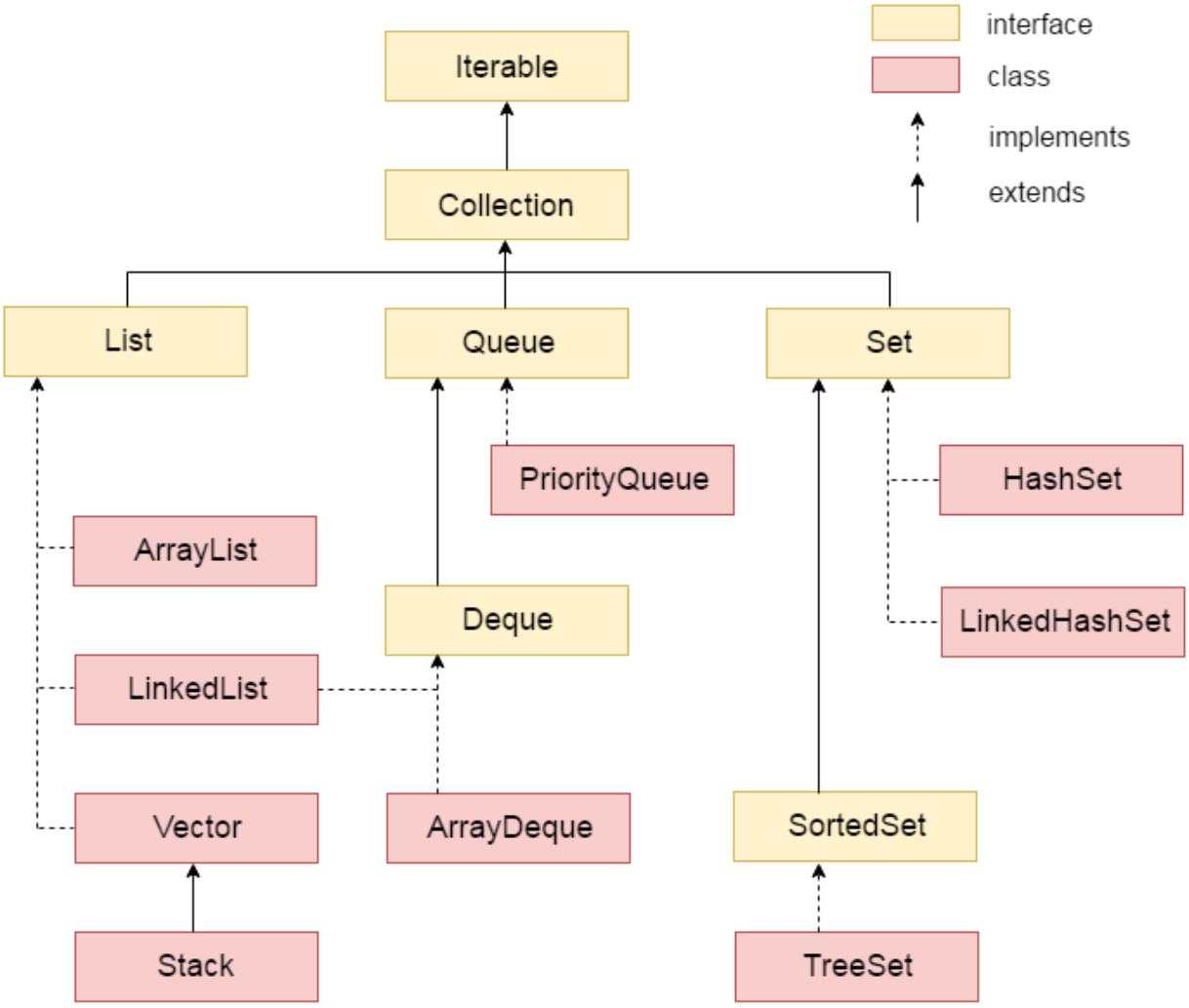
Collections in Java

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm



Iterator interface

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

Methods of Iterator interface

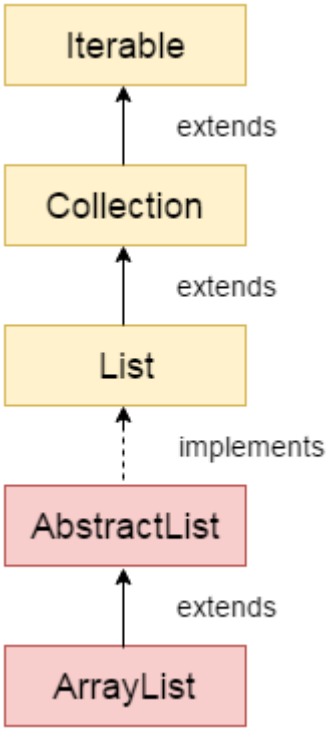
There are only three methods in the Iterator interface. They are:

1. **public boolean hasNext()** it returns true if iterator has more elements.
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 rarely used.

The important points about Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

As shown in above diagram, Java ArrayList class extends AbstractList class which implements List interface. The List interface extends Collection and Iterable interfaces in hierarchical order.



Let's see the old non-generic example of creating java collection.

```
ArrayList al=new ArrayList();//creating old non-generic arraylist
```

Let's see the new generic example of creating java collection.

```
ArrayList<String> al=new ArrayList<String>();//creating new generic arraylist
```

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

Two ways to iterate the elements of collection in java

There are two ways to traverse collection elements:

1. By Iterator interface.
2. By for-each loop.

```
import java.util.*;
class TestCollection1{
 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());   }  }
```

```
import java.util.*;
class TestCollection2{
 public static void main(String args[]){   ArrayList<String> al=new ArrayList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");
  for(String obj:al)
   System.out.println(obj);  } }
```

```
Ravi
Vijay
Ravi
Ajay
```

Let's see an example where we are storing Student class object in array list.

```
class Student{
  int rollno;
  String name;
  int age;
  Student(int rollno,String name,int age){
   this.rollno=rollno;
   this.name=name;
   this.age=age;
  }
}
```

```
import java.util.*;
public class TestCollection3{
 public static void main(String args[]){
  //Creating user-defined class objects
  Student s1=new Student(101,"Sonoo",23);
  Student s2=new Student(102,"Ravi",21);
  Student s2=new Student(103,"Hanumat",25);
  //creating arraylist
  ArrayList<Student> al=new ArrayList<Student>();
  al.add(s1);//adding Student class object
  al.add(s2);
  al.add(s3);
  //Getting Iterator
  Iterator itr=al.iterator();
  //traversing elements of ArrayList object
  while(itr.hasNext()){
   Student st=(Student)itr.next();
   System.out.println(st.rollno+" "+st.name+" "+st.age);   }  }  }
```

```
import java.util.*;
class TestCollection4{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String
>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ajay");
  ArrayList<String> al2=new ArrayList<Strin
g>();
  al2.add("Sonoo");
  al2.add("Hanumat");
  al.addAll(al2);//adding second list in first list

 Iterator itr=al.iterator();
 while(itr.hasNext()){
   System.out.println(itr.next());    }   } }
```

```
Ravi
Vijay
Ajay
Sonoo
Hanumat
```

Java LinkedList class

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
- Java LinkedList class can be used as list, stack or queue.

Doubly Linked List

In case of doubly linked list, we can add or remove elements from both side.

Let's see the declaration for java.util.LinkedList class.

public class LinkedList<E> **extends** AbstractSequentialList<E> **implements** List<E>, De que<E>, Cloneable, Serializable

```
import java.util.*;
public class TestCollection7{
 public static void main(String args[]){
  LinkedList<String> al=new LinkedList<Stri
ng>();
  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());    }  } }
```

```
Output:Ravi
       Vijay
       Ravi
       Ajay
```

ArrayList	LinkedList
1) ArrayList internally uses **dynamic array** to store the elements.	LinkedList internally uses **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
3) ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

Java List Interface

public interface List<E> extends Collection<E>

```
import java.util.*;
public class ListExample{
public static void main(String args[]){
ArrayList<String> al=new ArrayList<String>();
al.add("Amit"); al.add("Vijay"); al.add("Kumar"); al.a
dd(1,"Sachin");
System.out.println("Element at 2nd position: "+al.get(2
)); for(String s:al){  System.out.println(s);  }  }  }
```

```
Element at 2nd position:
Vijay
Amit
Sachin
Vijay
Kumar
```

Java ListIterator Interface

ListIterator Interface is used to traverse the element in backward and forward direction.

public interface ListIterator<E> **extends** Iterator<E>

```
import java.util.*;
public class TestCollection8{
public static void main(String args[]){
ArrayList<String> al=new ArrayList<String>();
al.add("Amit");
al.add("Vijay");
al.add("Kumar");
al.add(1,"Sachin");
System.out.println("element at 2nd position: "+al.get(2))
;
. ListIterator<String> itr=al.listIterator();
. System.out.println("traversing elements in forward direct
ion...");
. while(itr.hasNext()){
. System.out.println(itr.next());  }
. System.out.println("traversing elements in backward dire
ction...");
. while(itr.hasPrevious()){  System.out.println(itr.previous
()); }  }  }
```

```
element at 2nd position:
Vijay
traversing elements in
forward direction...
Amit
Sachin
Vijay
Kumar
traversing elements in
backward direction...
Kumar
Vijay
Sachin
Amit
```

Java HashSet class

The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing.**
- HashSet contains unique elements only.

Difference between List and Set

List can contain duplicate elements whereas Set contains unique elements only.

HashSet class declaration

Let's see the declaration for java.util.HashSet class.

public class HashSet<E> **extends** AbstractSet<E> **implements** Set<E>, Cloneable, Seria
lizable