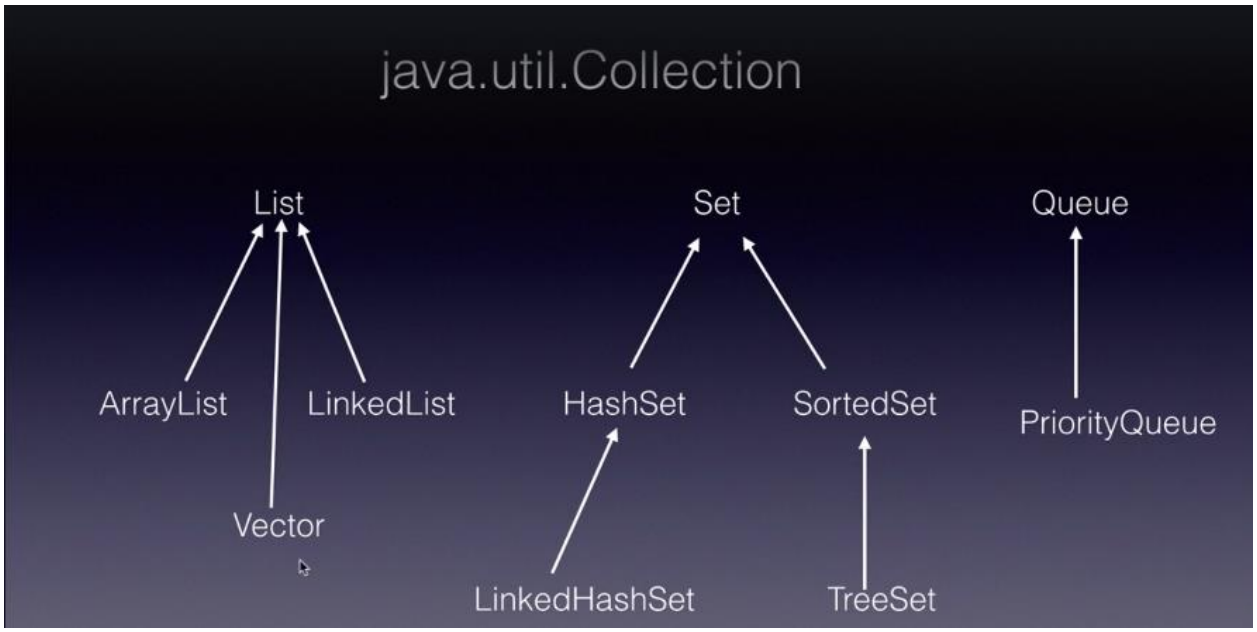
Arrays

- *Arrays are used to represent group of elements as a single entity but these elements are **homogeneous** & **fixed** size.*
- *The size of Array is fixed it means once we created Array it is not possible to increase and decrease the size.*
- ***Array in java is index based first element of the array stored at 0 index.***
- ***Advantages of array:-***
- Instead of declaring individual variables we can declare group of elements by using array it reduces length of the code.
- We can store the group of objects easily & we are able to retrieve the data easily.
- We can access the random elements present in the any location based on index.

declaration & instantiation & initialization :-

- ***Approach 1:-*** int a[]={10,20,30,40}; ***//declaring, instantiation, intialization***
- ***Approach 2:-*** int[] a=new int[100]; ***//declaring, instantiation***
- a[0]=10; ***//initialization***
- a[1]=20;
- ;;;;;;;;;;;;;;;;;;;;;
- a[99]=40;
- ***// declares an array of integers***
- int[] anArray;
- ***// allocates memory for 10 integers***
- anArray = new int[10];
- ***// initialize first element***
- anArray[0] = 10;
- ***// initialize second element***
- anArray[1] = 20;
- ***// and so forth***
- anArray[2] = 30; anArray[3] = 40; anArray[4] = 50; anArray[5] = 60;
- anArray[6] = 70; anArray[7] = 80; anArray[8] = 90; anArray[9] = 100;

Collection(Interface)



Collection vs Collections:-

Collection is interface it is used to represent group of objects as a single entity.
Collections is utility class it contains methods to perform operations.

Note: - The root interface of Collection framework is **Collection** it contains 15 methods so all Implementation classes are able to use that methods.

- public abstract int size();
- public abstract boolean isEmpty();
- public abstract boolean contains(java.lang.Object);
- public abstract java/util/Iterator<E> iterator();
- public abstract java.lang.Object[] toArray();
- public abstract <T extends java/lang/Object> T[] toArray(T[]);
- public abstract boolean add(E);
- public abstract boolean remove(java.lang.Object);
- public abstract boolean containsAll(java/util/Collection<?>);
- public abstract boolean addAll(java/util/Collection<? extends E>);
- public abstract boolean removeAll(java/util/Collection<?>);
- public abstract boolean retainAll(java/util/Collection<?>);
- public abstract void clear();

- public abstract boolean equals(java.lang.Object);
- public abstract int hashCode();

Characteristics of Collection frame work classes:-

- *The collect ion framework classes are introduced in different Versions.*
- *Heterogeneous data allowed or not allowed.*
  - *Null insertion is possible or not possible.*
- *Insertion order is preserved or not preserved.*
- *Input --->e1 e2 e3 output --->e1 e2 e3 insertion order is preserved*
- *Input --->e1 e2 e3 output --->e2 e1 e3 insertion order is not-preserved*
- *Collection classes' methods are synchronized or non-synchronized.*
- *Duplicate objects are allowed or not allowed.*
- *Collections classes **underlying data structures**.*

Implementation classes of List interface:-

1) ArrayList 2) LinkedList 3) Vector (Legacy) 4) Stack

Java.util.ArrayList:-

*ArrayList is **implementing** List interface it widely used class in projects because it is providing functionality and flexibility.*

ArrayList Characteristics:-

- ArrayList Introduced in **1.2** version.
- ArrayList stores Heterogeneous objects (different types).
- Inside ArrayList we can insert **Null** objects.
- ArrayList preserved Insertion order it means whatever the order we inserted the data in the same way output is printed.
- ArrayList methods are non-synchronized methods.
- Duplicate objects are allowed.
- The under laying data structure is growable array.

Constructors

- *ArrayList al = new ArrayList(); (Default capacity - 10).*
- *ArrayList al = new ArrayList (int initial-capacity)*
- *ArrayList al = new ArrayList(Collection c);*

Example :-Collections Autoboxing

upto 1.4 version by using wrapper classes, create objects then add that objects in ArrayList.

```
import java.util.ArrayList;
class Test
{ public static void main(String[] args)
{ ArrayList al = new ArrayList();
Integer i = new Integer(10); //creation of Integer Object
Character ch = new Character('c'); //creation of Character Object
Double d = new Double(10.5); //creation of Double Object
//adding wrapper objects into ArrayList
al.add(i);
al.add(ch);
al.add(d);
System.out.println(al);
                }
```

But from 1.5 version onwards *autoboxing concept is introduced so add the primitive value directly that is automatically converted into wrapper objects format.*

```
import java.util.ArrayList;
class Test
{ public static void main(String[] args)
{ ArrayList al = new ArrayList();
al.add(10); //primitive int value --->Integer Object conversion //AutoBoxing
al.add('a'); //primitive char value --->Integer Object conversion //AutoBoxing
al.add(10.5); //primitive double value --->Integer Object conversion //AutoBoxing
System.out.println(al);
}
}
```

Methods of ArrayList

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified p
boolean add(E e)	It is used to append the specified element at the end of a
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified col this list, in the order that they are returned by the specifie iterator.
boolean addAll(int index, Collection<?	It is used to append all the elements in the specified collec

extends E> c)	specified position of the list.
void clear()	It is used to remove all of the elements from this list.
E get(int index)	It is used to fetch the element from the particular position
boolean isEmpty()	It returns true if the list is empty, otherwise false.

Java Non-generic Vs. Generic Collection

Java collection framework **was non-generic before JDK 1.5**. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in a collection. Now it is type safe so typecasting is not required at runtime.

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

- ArrayList list=**new** ArrayList();//creating old non-generic arraylist
- ArrayList<String> list=**new** ArrayList<String>();//creating new generic arraylist
- In a generic collection, we specify the type in angular braces. Now ArrayList is forced to have the only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

Example code :

```
import java.util.*;
 public class ArrayListExample1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
      list.add("Java");//Adding object in arraylist
      list.add("Spring");
      list.add("SpringBoot");
      list.add("Hibernate");
      //Printing the arraylist object
      System.out.println(list);
 }
}
```

Iterating ArrayList using Iterator

```
1.  import java.util.*;
public class ArrayListExample2{
```

```
public static void main(String args[]){
 ArrayList<String> list=new ArrayList<String>();//Creating arraylist
 list.add("C");//Adding object in arraylist
 list.add("Java");
 list.add("Python");
 list.add("Spring");
 //Traversing list through Iterator
 Iterator itr=list.iterator();//getting the Iterator
 while(itr.hasNext()){//check if iterator has the elements
  System.out.println(itr.next());//printing the element and move to next
 }
 }
}
```

- Iterating ArrayList using For-each loop

Sort ArrayList

- Collections.sort(sourceList)

User-defined class objects in Java ArrayList

Java LinkedList class

Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- 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 occur**.

Doubly Linked List

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

fig- doubly linked list

Constructors

- LinkedList()
- LinkedList(Collection<? extends E> c)
- Methods similar to Array list an additional methods of linked list will be present.

ArrayList	LinkedList
1) ArrayList internally uses a **dynamic array** to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses an 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 a doubly linked list, so no bit shifting is required in memory.
3) An 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.

Stack:- (legacy class introduced in 1.0 version)
1) It is a child class of **vector**
2) Introduce in 1.0 v legacy class
3) It is designed for LIFO(last in fist order)

Example:-

```
import java.util.*;
class Test
{ public static void main(String[] args)
{ Stack<String> s = new Stack<String>();
s.push("ratan"); //insert the data top of the stack
s.push("anu"); //insert the data top of the stack
s.push("Sravya");
```

```
System.out.println(s);
System.out.println(s.size());
System.out.println(s.peek()); //to return last element of the Stack
s.pop(); //remove the data top of the stack
System.out.println(s);
System.out.println(s.isEmpty());
s.clear();
System.out.println(s.isEmpty());
}
}
```

HashSet:-

1) Introduced in 1.2 v.
2) Duplicate objects are not allowed if we are trying to insert duplicate values then we won't get any compilation errors an won't get any Execution errors simply add method return old value.
3) Null insertion is possible but if we are inserting more than one null it return only one null value.
4) Heterogeneous objects are allowed.
5) The under laying data structure is HashTable.
6) It is not maintain any order the elements are return in any random order .[Insertion order is not
preserved].
7) Methods are non-synchronized.
8) cursor : Iterator.

Linked HashSet and TreeSet methods same with different behaviours.