Object - Oriented Programming 2

Week 11. Iterator and ListIterator, ArrayList Class, LinkedList Class, HashSet Class and PriorityQueue Class

By Elubu Joseph - MSc.IS

Lecturer

Department of Information Technology

Kumi University

Email: josebulinda@gmail.com

jose@kumiuniversity.ac.ug

Agenda

1. Iterator and ListIterator
2. ArrayList Class,
3. LinkedList Class,
4. HashSet Class and
5. PriorityQueue

Iterator and ListIterator

Iterator is an interface that is used to iterate the collection elements. It is part of java collection framework. It provides some methods that are used to check and access elements of a collection.

Iterator Interface is used to traverse a list in forward direction, enabling you to remove or modify the elements of the collection. Each collection classes provide iterator() method to return an iterator.

Accessing a Java Collection using Iterators

To access elements of a collection, we can either use index if collection is list based or we need to traverse the element. There are three possible ways to traverse through the elements of any collection.

1. Using Iterator interface
2. Using ListIterator interface
3. Using for-each loop

Iterator Interface Methods

Method	**Description**
boolean hasNext()	Returns **true** if there are more elements in the collection. Otherwise, returns false.
E next()	Returns the **next element present** in the collection. Throws NoSuchElementException if there is not a next element.
void remove()	Removes the current element. Throws IllegalStateException if an attempt is made to call remove() method that is not preceded by a call to next() method.

Iterator Example Program

In this example, we are using iterator() method of collection interface that returns an instance of Iterator interface. After that we are using hasNext() method that returns true if collection contains elements and within the loop, obtain each element by calling next() method.

```
import java.util.*;
class ITPro {
public static void main(String[] args) {
      ArrayList< String> al = new ArrayList< String>();
      al.add("BIT");
      al.add("CIT");
      al.add("ET");
      al.add("DIT");
      Iterator it = al.iterator(); //Declaring Iterator
      while(it.hasNext()) {
      System.out.print(it.next()+" ");
      }
} }
```

Iterator Example Program code+output

```
1  package CollectionClass; import java.util.*;
2  class ITPro {
3      public static void main(String[] args) {
           ArrayList< String> al = new ArrayList< String>();
5          al.add("BIT");
6          al.add("CIT");
7          al.add("ET");
8          al.add("DIT");
9          Iterator it = al.iterator(); //Declaring Iterator
10         while(it.hasNext()) {
11         System.out.print(it.next()+" ");
12         }
13     } }
```

```
run:
BIT CIT ET DIT
```

ListIterator

Accessing elements using ListIterator

ListIterator Interface is used to traverse a list in both forward and backward direction. It is available to only those collections that implements the ListInterface.

Methods of ListIterator

Method	Description
void **add**(E obj)	Inserts obj into the list in front of the element that will be returned by the next call to next() method.
boolean **hasNext**()	Returns true if there is a next element. Otherwise, returns false.
boolean **hasPrevious**()	Returns true if there is a previous element. Otherwise, returns false.
E **next**()	Returns the next element. A NoSuchElementException is thrown if there is not a next element.
int **nextIndex**()	Returns the index of the next element. If there is not a next element, returns the size of the list.

Methods of ListIterator+

Method	Description
E **previous**()	Returns the previous element. A NoSuchElementException is thrown if there is not a previous element.
int **previousIndex**()	Returns the index of the previous element. If there is not a previous element, returns -1.
void **remove**()	Removes the current element from the list. An IllegalStateException is thrown if remove() method is called before next() or previous() method is invoked.
void **set**(E obj)	Assigns obj to the current element. This is the element last returned by a call to either next() or previous() method.

ListIterator Example

In this example, we are creating a program to traverse the elements of ArrayList using ListIterator. ListIterator works only with list collection.

```
import java.util.*;
class ILTraverse {
public static void main(String[] args) {
      ArrayList< String> al = new ArrayList< String>();
      al.add("PEM"); al.add("CRE"); al.add("PHY"); al.add("CHEM");
      ListIterator lob = al.listIterator();
      while(lob.hasNext()){//In forward direction
            System.out.print(lob.next()+" ");
      }
       System.out.println("Backward Traverse");
      while(lob.hasPrevious()) {//In backward direction
            System.out.print(lob.previous()+" ");
      }
} }
```

ListIterator Example code and output

```
1  package CollectionClass;import java.util.*;
2  class ILTraverse {
3  public static void main(String[] args) {
4          ArrayList< String> al = new ArrayList< String>();
5          al.add("PEM"); al.add("CRE"); al.add("PHY");
6          al.add("CHEM");
7          ListIterator lob = al.listIterator();
8          while(lob.hasNext()){//In forward direction
9                  System.out.print(lob.next()+" ");
10         }
11         System.out.println("\nBackward Traverse");
12         while(lob.hasPrevious()) {//In backward direction
13                 System.out.print(lob.previous()+" ");
14         }
15 } }
```

```
run:
PEM CRE PHY CHEM
Backward Traverse
CHEM PHY CRE PEM
```

Collection Framework ArrayList

Collection Framework ArrayList

This class provides implementation of an array based data structure that is used to store elements in linear order. This class implements List interface and an abstract AbstractList class. It creates a dynamic array that grows based on the elements strength.

Note!

We covered much about the class in lecture 10, we will therefore focus on the areas we did not handle.

ArrayList Methods

The table below contains methods of Arraylist. We can use them to manipulate its elements.

Method	Description
void **add**(int index, E element)	inserts the specified element at the specified position in a list.
boolean **add**(E e)	appends the specified element at the end of a list.
boolean **addAll**(Collection<? extends E> c)	appends all of the elements in the specified collection to the end of this list.
boolean **addAll**(int index, Collection<? extends E> c)	appends all the elements in the specified collection, starting at the specified position of the list.
void **clear**()	removes all of the elements from this list.
void **ensureCapacity**(int requiredCapacity)	enhances the capacity of an ArrayList instance.
E **get**(int index)	fetches the element from the particular position of the list.
boolean **isEmpty**()	returns true if the list is empty, otherwise false.

ArrayList Methods+

Method	Description
int **lastIndexOf**(Object o)	returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] **toArray**()	returns an array containing all of the elements in this list in the correct order.
<T> T[] **toArray**(T[] a)	returns an array containing all of the elements in this list in the correct order.
Object **clone**()	returns a shallow copy of an ArrayList.
boolean **contains**(Object o)	returns true if the list contains the specified element
int **indexOf**(Object o)	returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.

ArrayList Methods++

Method	Description
E **remove**(int index)	removes the element present at the specified position in the list.
boolean **remove**(Object o)	removes the first occurrence of the specified element.
boolean **removeAll**(Collection<?> c)	removes all the elements from the list.
boolean **removeIf**(Predicate<? super E> filter)	removes all the elements from the list that satisfies the given predicate.
protected void **removeRange**(int fromIndex, int toIndex)	removes all the elements lies within the given range.
void **replaceAll**(UnaryOperator<E> operator)	replaces all the elements from the list with the specified element.
void **trimToSize**()	trims the capacity of this ArrayList instance to be the list's current size.

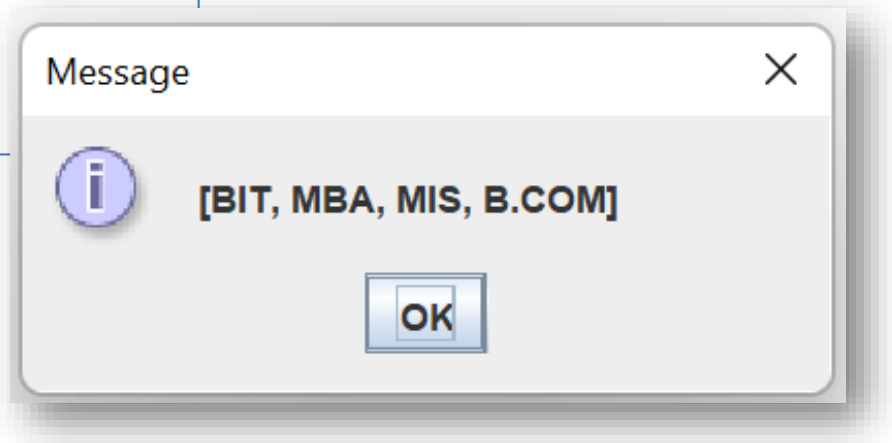
Adding items into ArrayList Example program

In this example, we added items into ArrayList to store string elements., we used add method of list interface to add elements.

```
import java.util.*; import javax.swing.JOptionPane;
class AList {
      public static void main(String[] args) {
      ArrayList< String> al = new ArrayList< String>();
      al.add("BIT");
      al.add("MBA");
      al.add("MIS");
      al.add("B.COM");
      JOptionPane.showMessageDialog(null, al);
} }
```

ArrayList Class++++ Program Code+Output

```
1   package CollectionClass;
2   import java.util.*; import javax.swing.JOptionPane;
3   public class AList {
4       public static void main(String[] args) {
            ArrayList< String> al = new ArrayList< String>();
6           al.add("BIT");
7           al.add("MBA");
8           al.add("MIS");
9           al.add("B.COM");
10          JOptionPane.showMessageDialog(null, al);
11      }
12  }
```



Removing Elements from ArrayList

To remove elements from the list, we use remove() method that removes the specified elements. We can also pass index value to remove the elements of it.

```
import java.util.*;
class RAList {
	public static void main(String[] args) {
	ArrayList< String> al = new ArrayList< String>();
	al.add("BIT");
	al.add("MBA");
	al.add("MIS");
	al.add("B.COM");
	System.out.println("Inserted Elements\n"+ al);
	al.remove(3);
	System.out.println("After removing 1 Element\n"+ al);

} }
```

Removing Elements from ArrayList code and output

```
1   package CollectionClass;
2   import java.util.*; import javax.swing.JOptionPane;
3   class RAList {
4           public static void main(String[] args) {
5           ArrayList< String> al = new ArrayList< String>();
6           al.add("BIT");
7           al.add("MBA");
8           al.add("MIS");
9           al.add("B.COM");
10          System.out.println("Inserted Elements\n" + al);
11          al.remove(3);
12          System.out.println("After removing 1 Element\n"+ al);
13
14  } }
```

Note! Element at index 3 has been removed from the list.

```
run:
Inserted Elements
[BIT, MBA, MIS, B.COM]
After removing 1 Element
[BIT, MBA, MIS]
```

Get size of ArrayList

Sometimes we may want to know number of elements an ArrayList holds. In that case we use size() method that returns size of ArrayList which is equal to number of elements present in the list.

In this case, we will only add the following code into the code above.

```
for(String el : al) {
      System.out.println(el);
}
      System.out.println("Total Elements: "+al.size());
```

Getting ArrayList Size program

```
import java.util.*;
class AraySize {
	public static void main(String[] args) {
	ArrayList< String> al = new ArrayList< String>();
	al.add("BIT");
	al.add("MBA");
	al.add("MIS");
	al.add("B.COM");
	System.out.println("Inserted Elements\n"+ al);
	for(String el : al) {
		System.out.println(el);
	}
	System.out.println("Total Elements: "+al.size());

	al.remove(3);
	System.out.println("After removing 1 Element\n"+ al);
	 System.out.println("Total Elements: "+al.size());
} }
```

Getting ArrayList Size program code+output

```
1   package CollectionClass;
2   import java.util.*; import javax.swing.JOptionPane;
3   class AraySize {
4           public static void main(String[] args) {
5           ArrayList< String> al = new ArrayList< String>();
6           al.add("BIT"); al.add("MBA");
7           al.add("MIS"); al.add("B.COM");
8           System.out.println("Inserted Elements\n"+ al);
9           for(String el : al) {
10                  System.out.println(el);
11          }
12          System.out.println("Total Elements: "+al.size());
13          al.remove(3);
14          System.out.println("After removing 1 Element\n"+ al);
15          System.out.println("Total Elements: "+al.size());
16  } }
```

```
run:
Inserted Elements
[BIT, MBA, MIS, B.COM]
BIT
MBA
MIS
B.COM
Total Elements: 4
After removing 1 Element
[BIT, MBA, MIS]
Total Elements: 3
```

Sorting ArrayList Elements

To sort elements of an ArrayList, Java provides a class Collections that includes a static method sort(). In this example, we are using sort method to sort the elements by adding the following line of code in the program above.

```
Collections.sort(al);
// Traversing ArrayList
    for(String el : al) {
    System.out.println(el);
    }
```

Assignment. Complete the above program by adding the above code.

LinkedList Class

LinkedList class

Java LinkedList class provides implementation of **linked-list data structure**. It used doubly linked list to store the elements. It implements List, Deque and Queue interface and extends AbstractSequentialList class. We shall see the declaration of this class in the next slide but lets first look at its features.

Features

1. LinkedList class extends AbstractSequentialList and implements **List**, **Deque** and **Queue** interface.
2. It can be used as List, stack or Queue as it implements all the related interfaces.
3. It is dynamic in nature i.e it allocates memory when required. Therefore insertion and deletion operations can be easily implemented.
4. It can contain duplicate elements and it is not synchronized.
5. Reverse Traversing is difficult in linked list.
6. In LinkedList, manipulation is fast because no shifting needs to be occurred.

LinkedList class Declaration and Constructors

LinkedList is declared as below.

```
public class LinkedList<E> extends AbstractSequentialList<E>
implements List<E>, Deque<E>, Cloneable, Serializable
```

LinkedList class has two constructors.

```
LinkedList() // It creates an empty LinkedList
LinkedList( Collection c)
```

Note. Like ArrayList, we will only cover some few things we did not handle in lecture 10.

LinkedList Methods

The below table contains methods of LinkedList. We can use them to manipulate its elements.

Method	Description
boolean **add**(E e)	It appends the specified element to the end of a list.
void **add**(int index, E element)	It inserts the specified element at the specified position index in a list.
boolean **addAll**(Collection<? extends E> c)	It appends all of the elements in the specified collection to the end of this list.
boolean **addAll**(Collection<? extends E> c)	It appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean **addAll**(int index, Collection<? extends E> c)	It appends all the elements in the specified collection, starting at the specified position of the list.
void **addFirst**(E e)	It inserts the given element at the beginning of a list.
void **addLast**(E e)	It appends the given element to the end of a list.

LinkedList Methods

The below table contains methods of LinkedList. We can use them to manipulate its elements.

Method	**Description**
void **clear**()	It removes all the elements from a list.
Object **clone**()	It returns a shallow copy of a list.
boolean **contains**(Object o)	It returns true if a list contains a specified element.
Iterator<E> descendingIterator()	It returns an iterator over the elements in a deque in reverse sequential order.
E **element**()	It retrieves the first element of a list.
E **get**(int index)	It returns the element at the specified position in a list.
E **getFirst**()	It returns the first element in a list.
E **getLast**()	It returns the last element in a list.
int **indexOf**(Object o)	It returns the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.

LinkedList Methods

The below table contains methods of LinkedList. We can use them to manipulate its elements.

Method	Description
int **lastIndexOf**(Object o)	is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.
ListIterator<E> **listIterator**(int index)	returns a list-iterator of the elements in proper sequence, starting at the specified position in the list.
boolean **offer**(E e)	adds the specified element as the last element of a list.
boolean **offerFirst**(E e)	inserts the specified element at the front of a list.
boolean **offerLast**(E e)	inserts the specified element at the end of a list.
E **peek**()	retrieves the first element of a list
E **peekFirst**()	retrieves the first element of a list or returns null if a list is empty.
E **peekLast**()	retrieves the last element of a list or returns null if a list is empty.
E **poll**()	retrieves and removes the first element of a list.
E **pollFirst**()	retrieves and removes the first element of a list, or returns null if a list is empty.

LinkedList Methods

The below table contains methods of LinkedList. We can use them to manipulate its elements.

Method	Description
E **pollLast**()	retrieves and removes the last element of a list, or returns null if a list is empty.
E **pop**()	pops an element from the stack represented by a list.
void **push**(E e)	pushes an element onto the stack represented by a list.
E **remove**()	is used to retrieve and removes the first element of a list.
E **remove**(int index)	is used to remove the element at the specified position in a list.
boolean **remove**(Object o)	is used to remove the first occurrence of the specified element in a list.
E **removeFirst**()	removes and returns the first element from a list.
boolean **removeFirstOccurrence**(Object o)	removes the first occurrence of the specified element in a list (when traversing the list from head to tail).
E **removeLast**()	removes and returns the last element from a list.
boolean **removeLastOccurrence**(Object o)	removes the last occurrence of the specified element in a list (when traversing the list from head to tail).
E **set**(int index, E element)	replaces the element at the specified position in a list with the specified element.
int **size**()	returns the number of elements in a list.

(studytonight.com)

Add and Remove Elements to LinkedList

To add elements into LinkedList, we used add() method. It adds elements into the list in the insertion order. Refer to lecture 10.

Removing Elements

To remove elements from the list, we use remove() method that remove the specified elements from the list as seen in ArrayList element removal above. We can also pass index value to remove the elements of it.

Get size of LinkedList

Sometimes we want to know number of elements an LinkedList holds. In that case we use size() then returns size of LinkedList which is equal to number of elements present in the list.

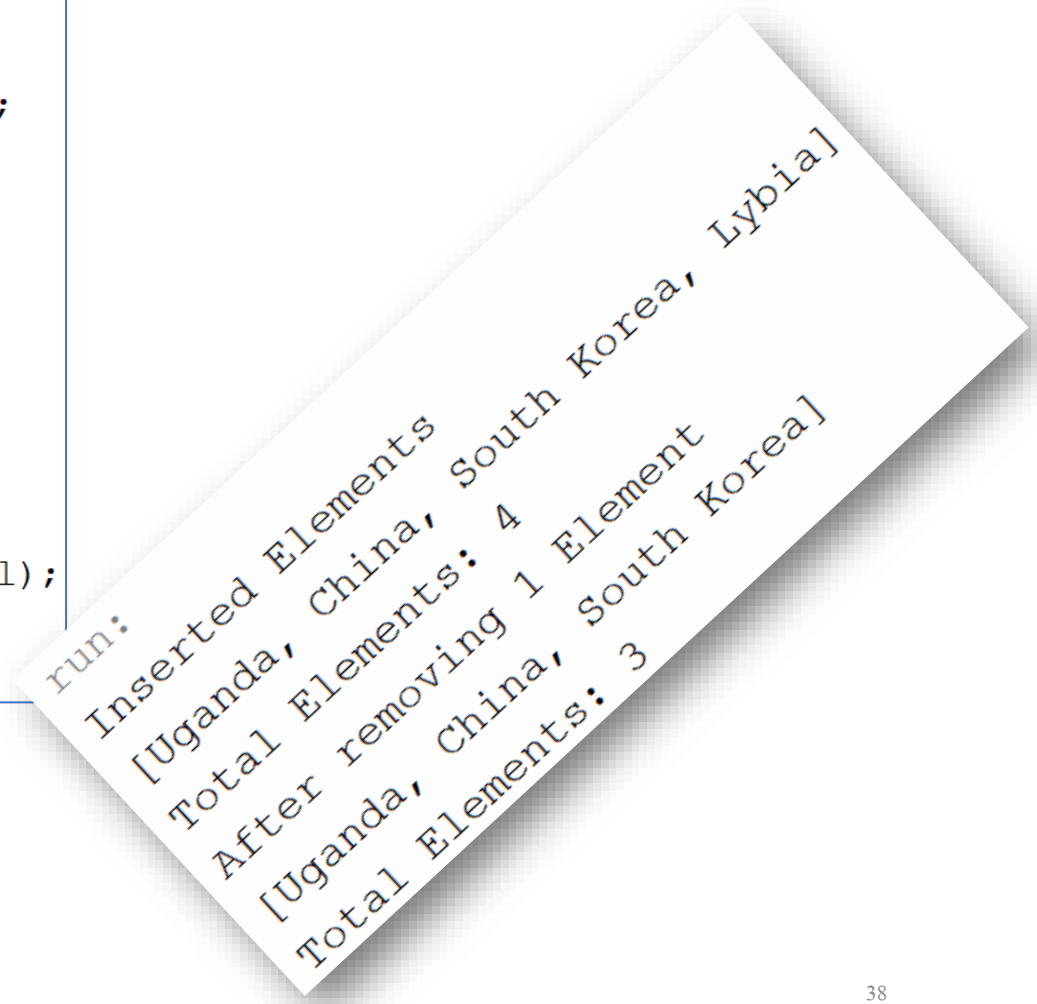
Just like we did in Arraylist Size extraction above, lets do it here.

Getting LinkedList Size program

```
import java.util.*;
class TCountries {
	public static void main(String[] args) {
	LinkedList< String> al = new LinkedList< String>();
	al.add("Uganda");
	al.add("China");
	al.add("South Korea");
	al.add("Lybia");
	System.out.println("Inserted Elements\n"+ al);

	System.out.println("Total Elements: "+al.size());

	al.remove(3);
	System.out.println("After removing 1 Element\n"+ al);
	System.out.println("Total Elements: "+al.size());

} }
```

Getting ArrayList Size program code and output

```
1  package CollectionClass; import java.util.*;
2  class TCountries {
3          public static void main(String[] args) {
4          LinkedList< String> al = new LinkedList< String>();
5          al.add("Uganda");
6          al.add("China");
7          al.add("South Korea");
8          al.add("Lybia");
9          System.out.println("Inserted Elements\n"+ al);
10
11         System.out.println("Total Elements: "+al.size());
12         al.remove(3);
13         System.out.println("After removing 1 Element\n"+ al);
14         System.out.println("Total Elements: "+al.size());
15 } }
```



Sorting LinkedList Elements

To sort elements of an LinkedList, Java provides a class Collections that includes a static method sort(). In this example, we are using sort method to sort the elements.

The lines of code below will help us sort or LinkedList in the program above.

```
Collections.sort(al); // Traversing LinkedList
for(String el : al){
     System.out.println(el);
}
```

Sorting LinkedList sample program code

```
import java.util.*;
class sortLList {
	public static void main(String[] args) {
	LinkedList< String> al = new LinkedList< String>();
	al.add("Uganda");
	al.add("China");
	al.add("South Korea");
	al.add("Lybia");
	Collections.sort(al); // Traversing LinkedList
	for(String el : al){
	System.out.println(el);
	}
} }
```

Sorting LinkedList sample program code and output

```
1   package CollectionClass;
2   import java.util.*;
3   class sortLList {
4           public static void main(String[] args) {
            LinkedList< String> al = new LinkedList< String>();
6           al.add("Uganda");
7           al.add("China");
8           al.add("South Korea");
9           al.add("Lybia");
10          Collections.sort(al); // Traversing LinkedList
11          for(String el : al){
12          System.out.println(el);
13          }
14  } }
```

```
run:
China
Lybia
South Korea
Uganda
```

LinkedList Traverse using for-each loop

for-each version of for loop can also be used for traversing the elements of a collection. But this can **only be used if we don't want to modify** the contents of a collection and **we don't want any reverse** access. **for-each loop** can cycle through any collection of object that implements Iterable interface.

Example program LinkedList Traverse using for-each loop

```
import java.util.*;
class forTraverse {
public static void main(String[] args) {
      LinkedList< String> al = new LinkedList< String>();
      al.add("PEM"); al.add("CRE"); al.add("PHY");
      al.add("CHEM");

      for(String st : al) {
            System.out.print(st+" ");
      }
}
}
```

Example program LinkedList Traverse using for-each loop code and output

```
1   package CollectionClass;
2   import java.util.*;
3   class forTraverse {
4   public static void main(String[] args) {
5           LinkedList< String> al = new LinkedList< String>();
6           al.add("PEM"); al.add("CRE"); al.add("PHY"); al.add("CHEM");
7
8           for(String st : al) {
9                   System.out.print(st+" ");
10          }
11
12  }
13  }
```

```
run:
PEM CRE PHY CHEM
```

Example program LinkedList Traverse using normal for loop code and output

```
import java.util.*;
class forTraverse {
public static void main(String[] args) {
	LinkedList< String> al = new LinkedList< String>();
	al.add("LOVE"); al.add("HATE"); al.add("COME"); al.add("GO");

	for(int test = 0; test<al.size(); test++) {
		System.out.print(al+" ");
	}

}
}
```

Example program LinkedList Traverse using for loop code and output

```
1   package CollectionClass;
2   import java.util.*;
3   class forTraverse {
4   public static void main(String[] args) {
            LinkedList< String> al = new LinkedList< String>();
6           al.add("LOVE"); al.add("HATE");
7           al.add("COME"); al.add("GO");
8
            for(int test = 0; test<al.size(); test++ ) {
10                  System.out.print(al+" "+"\n");
11          }
12  }
13  }
```

```
run:
[LOVE, HATE, COME, GO]
[LOVE, HATE, COME, GO]
[LOVE, HATE, COME, GO]
[LOVE, HATE, COME, GO]
```

Note! Total number of items in the list means the number of rounds to loop.

PriorityQueue Class

Priority Queue

is an abstract data type that is similar to a queue, and every element has some priority value associated with it.

The priority of the elements in a priority queue determines the order in which elements are served (i.e., the order in which they are removed). If in any case the elements have same priority, they are served as per their ordering in the queue.

PriorityQueue Class +

Important features

1.extends the **AbstractQueue** class.

2.provides the facility of using queue.

3.It does not order the elements in FIFO manner.

PriorityQueue Class Constructors

PriorityQueue has six constructors.

In all cases, the capacity grows automatically as elements are added.

```
PriorityQueue( ) //creates an empty queue. By default, its starting
capacity is 11
PriorityQueue(int capacity) //creates a queue that has the specified
initial capacity
PriorityQueue(int capacity, Comparator comp) //creates a queue with the
specified capacity and comparator
//The last three constructors create queues that are initialized with
elements of Collection passed in c
PriorityQueue(Collection c)
PriorityQueue(PriorityQueue c)
PriorityQueue(SortedSet c)
```

PriorityQueue Class Constructors +

Note: If no comparator is specified when a PriorityQueue is constructed, then the default comparator for the type of data stored in the queue is used.

The default comparator will order the queue in ascending order. Thus, the head of the queue will be the smallest value. However, by providing a custom comparator, you can specify a different ordering scheme.

PriorityQueue class example program

In this example we are creating a **Priority Queue** that store and remove elements.

```
Import java.util.*;
class PQueue{
public static void main(String args[]) {
      PriorityQueue<String> qe=new PriorityQueue<String>();
      qe.add("MOVE"); qe.add("FORWARD"); qe.add("WITH");
      qe.add("STUDIES");
      System.out.println("Queue Home:"+qe.element());
      System.out.println("Queue End:"+qe.peek());
      System.out.println("Looping queue elements:");
      Iterator loop = qe.iterator();
      while(loop.hasNext()){
      System.out.println(loop.next()); }
      qe.remove(); qe.poll();
      System.out.println("After removing two elements:");
      Iterator loopa =qe.iterator();
       while(loopa.hasNext()){
      System.out.println(loopa.next()); }
} }
```

PriorityQueue Class Constructors +

```
1  package CollectionClass; import java.util.*;
2  class PQueue{
3  public static void main(String args[]) {
4          PriorityQueue<String> qe=new PriorityQueue<String>();
5          qe.add("MOVE"); qe.add("FORWARD"); qe.add("WITH");
6          qe.add("STUDIES");
7          System.out.println("Queue Home:"+qe.element());
8          System.out.println("Queue End:"+qe.peek());
9          System.out.println("Looping queue elements:");
10         Iterator loop = qe.iterator();
11         while(loop.hasNext()){
12         System.out.println(loop.next()); }
13         qe.remove(); qe.poll();
14         System.out.println("After removing two elements:");
15         Iterator loopa =qe.iterator();
16          while(loopa.hasNext()){
17         System.out.println(loopa.next());
18 } } }
```

Note! Priority given to queue elements based below.

```
run:
Queue Home:FORWARD
Queue End:FORWARD
Looping queue elements:
FORWARD
MOVE
WITH
STUDIES
After removing two elements:
STUDIES
WITH
```

HashSet Class

Java HashSet class is used to store unique elements. It uses hash table internally to store the elements. It implements Set interface and extends the AbstractSet class. Declaration of the HashSet class is given below.

Decalration

```
public class HashSet<E>extends AbstractSet<E>implements Set<E>,
Cloneable, Serializable
```

HashSet Class+

Important Points is that HashSet Class:

1. creates a collection that uses hash table for storage. A hash table stores information by using a mechanism called hashing.
2. HashSet does not maintain any order of elements.
3. HashSet contains only unique elements.
4. allows to store null value.
5. is non synchronized.
6. is the best approach for search operations.
7. The initial default capacity of HashSet is 16. (studytonight.com)

HashSet Methods

Method	Description
add(E e)	It adds the specified element to this set if it is not already present.
clear()	It removes all of the elements from the set.
clone()	It returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
contains(Object o)	It returns true if this set contains the specified element.

HashSet Methods

Method	Description
isEmpty()	It returns true if this set contains no elements.
iterator()	It returns an iterator over the elements in this set.
remove(Object o)	It removes the specified element from this set if it is present.
size()	It returns the number of elements in the set.
spliterator()	It creates a late-binding and fail-fast Spliterator over the elements in the set.

(studytonwight.com)

Add to and Remove Elements from HashSet

In this example, we are creating a HashSet that store string values and remove some. Since HashSet does not store duplicate elements, we tried to add a duplicate elements but the output contains only unique elements.

Add to and Remove Elements from HashSet **sample program**

```
import java.util.*;
class AddRemoveHSet {
public static void main(String args[]){ // Creating HashSet
      HashSet<String> hs = new HashSet<String>();
      hs.add("Plants"); hs.add("Humans"); hs.add("Clouds");
      hs.add("Plants"); hs.add("Wind"); hs.add("Water"); //
      System.out.println(hs);
      hs.remove("Humans");
      System.out.println ("After Removing 1 Element: \n"+hs);
}
}
```

Add to and Remove Elements from HashSet sample program code and output

```
1   package CollectionClass;
2   import java.util.*;
3   class AddRemoveHSet {
4   public static void main(String args[]){ // Creating HashSet
5           HashSet<String> hs = new HashSet<String>();
6           hs.add("Plants"); hs.add("Humans"); hs.add("Clouds");
7           hs.add("Plants"); hs.add("Wind"); hs.add("Water"); //
8           System.out.println(hs);
9           hs.remove("Humans");
10          System.out.println ("After Removing 1 Element: \n"+hs);
11  }
12  }
```

```
run:
[Water, Wind, Clouds, Humans, Plants]
After Removing 1 Element:
[Water, Wind, Clouds, Plants]
```

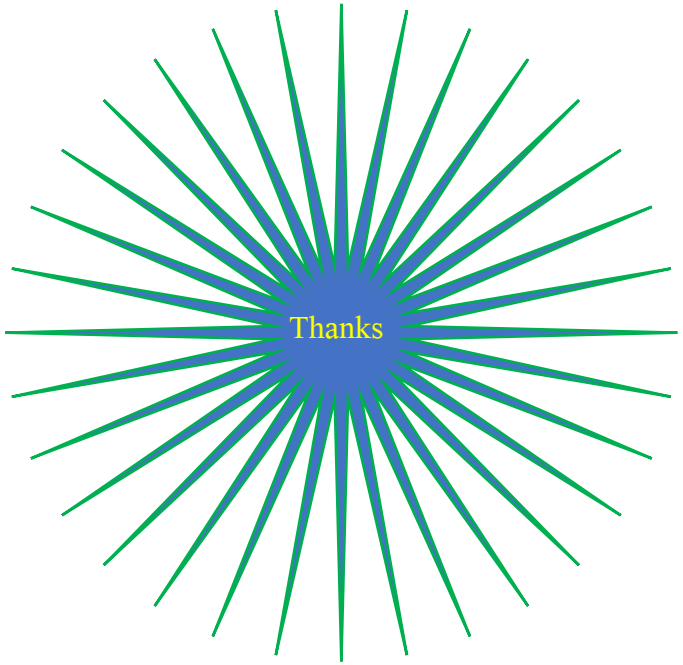
Assignment

1. Write a program that returns the size of a HashSet.
2. Modify program created in 1 above to traverse through the elements using for-each loop.

Summary

1. Iterator and ListIterator; had a recap on Iterator then looked at ListIterator(How it can be used in dealing with List Elements.) etc.
2. ArrayList Class looked at methods, important features and sampled some few methods,
3. LinkedList Class; definition, looked at methods, important features etc.
4. HashSet Class; definition, declaration, methods, use of add(), remove etc.
5. PriorityQueue; declaration, key teams and features, creating priority queue, methods, iteration of elements etc.

Thank you for Listening



References

Accessing a Java collection using iterators. Studytonight.com. (n.d.). Retrieved November 9, 2022, from https://www.studytonight.com/java/iterator-in-collection.php

Java Collection Framework Arraylist. Studytonight.com. (n.d.). Retrieved November 9, 2022, from https://www.studytonight.com/java/arraylist-in-collection-framework.php

Java Collection Framework Hashset. Studytonight.com. (n.d.). Retrieved November 8, 2022, from https://www.studytonight.com/java/hashset-in-collection-framework.php

Java Collection Framework Linked List. Studytonight.com. (n.d.). Retrieved November 7, 2022, from https://www.studytonight.com/java/linkedlist-in-collection-framework.php