AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

CHAPTER 10
LIST, STACKS, QUEUES AND PRIORITY QUEUES

SUBJECT:OOP-I CODE:3140705	PREPARED BY: ASST.PROF.NENSI KANSAGARA (CSE DEPARTMENT,ACET)	AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

COLLECTION OVERVIEW

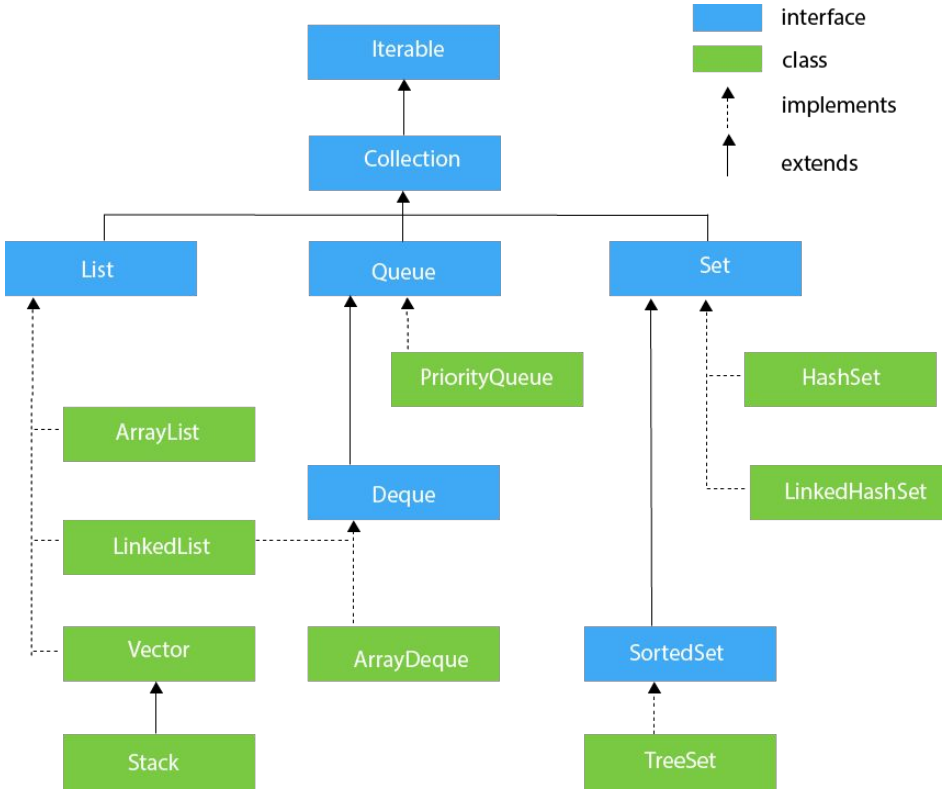
- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

WHAT IS A COLLECTION FRAMEWORK?

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

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

HIERARCHY OF COLLECTION FRAMEWORK



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

METHODS OF COLLECTION INTERFACE

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.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

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.

LIST INTERFACE

List Interface is the subinterface of Collection. It contains index-based methods to insert and delete elements. It is a factory of ListIterator interface.

List Interface declaration

public interface List<E> **extends** Collection<E>

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add(E e)	It is used to append the specified element at the end of a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of a list.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specifie position of the list.
void clear()	It is used to remove all of the elements from this list.
boolean equals(Object o)	It is used to compare the specified object with the elements of a list.
int hashcode()	It is used to return the hash code value for a list.
E get(int index)	It is used to fetch the element from the particular position of the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.
int lastIndexOf(Object o)	It is used to return 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()	It is used to return an array containing all of the elements in this list in the correct order.

T[] toArray(T[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean contains(Object o)	It returns true if the list contains the specified element
boolean containsAll(Collection<?> c)	It returns true if the list contains all the specified element
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element or -1 if the List does not contain this element.
E remove(int index)	It is used to remove the element present at the specified position in the list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element.
boolean removeAll(Collection<?> c)	It is used to remove all the elements from the list.
void replaceAll(UnaryOperator operator)	It is used to replace all the elements from the list with the specified element.
void retainAll(Collection<?> c)	It is used to retain all the elements in the list that are present in the specified collectio
E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
void sort(Comparator<? super E> c)	It is used to sort the elements of the list on the basis of specified comparator.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

SET INTERFACE

The set interface is used to define the set of elements.It extends the collection interface.This interface defined unique elements.hence if any duplicate elements in tried to insert in the set then the add() method return itself.

SORTEDSET INTERFACE

- A set is used to provide a particular ordering on its element. The elements are ordered either by using a natural ordering or by using a Comparator. All the elements which are inserted into a sorted set must implement the Comparable interface.
- The set's iterator will traverse the set in an ascending order. Several other operations are provided in order to make best use of ordering. All the elements must be mutually comparable.

Methods

comparator()	Returns the comparator which is used to order the elements in the given set. Also returns null if the given set uses the natural ordering of the element.
first()	Returns the first element from the current set.
headSet(E toElement)	Returns a view of the portion of the given set whose elements are strictly less than the toElement.
last()	Returns the reverse order view of the mapping which present in the map.
spliterator()	Returns a key-value mapping which is associated with the least key in the given map. Also, returns null if the map is empty.
subSet(E fromElement, E toElement)	Returns a key-value mapping which is associated with the greatest key which is less than or equal to the given key. Also, returns null if the map is empty.
tailSet(E fromElement)	Returns a view of the map whose keys are strictly less than the toKey.

MAP INTERFACE

This interface maps a unique key elements to the value.Thus map interface represent a key value pair.

SORTEDMAP INTERFACE

The SORTEDMAP is inherited from the Map interface.In this interface the elements are stored in ascending order.This Stored order is based on the key.

ITERATORS

An iterator is an interface that is used in place of Enumerations in the Java Collection Framework. Moreover, an iterator differs from the enumerations in two ways:

1. Iterator permits the caller to remove the given elements from the specified collection during the iteration of the elements.
2. Method names have been enhanced.

Iterator interface is a member connected with Java Collections Framework.

Methods

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.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

```
public class JavaApplicationIteratorProg {
    public static void main(String[] args) {
     ArrayList obj = new ArrayList();
     obj.add(10);
     obj.add(20);
     obj.add(30);
     obj.add(40);
     obj.add(50);

     System.out.println("The contents of array are:");
     Iterator itr_obj = obj.iterator();
     ListIterator Litr_obj=obj.listIterator();
     while(itr_obj.hasNext())
     {
         Object item=itr_obj.next();
         System.out.println("the item is" +item);
     }
     while(Litr_obj.hasNext())
     {
         Object item=Litr_obj.next();
     }
     System.out.println("The list in revrse order is:");
     while(Litr_obj.hasPrevious())
     {
         Object item = Litr_obj.previous();
         System.out.println("the item is" +item);
     }
    }
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

```
run:
The contents of array are:
the item is10
the item is20
the item is30
the item is40
the item is50
The list in revrse order is:
the item is50
the item is40
the item is30
the item is20
the item is10
BUILD SUCCESSFUL (total time: 1 second)
```

THE COMPARABLE INTERFACE

– – –

- The comparable interface is used to compare two objects of two different classes
- This interface is present in java.util.* package
- This interface defines two methods -

1.Compare()

2.equals()

SYNTAX:

public void sort(List list,Comparator c)

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Method	Description
Int Compare(Object obj1,Object obj2)	This method compares obj1 and obj2.It returns 0 if these objects are equal.It returns positive value if obj1>obj2.It returns negative if obj1<obj2
boolean equals(Object obj)	The object is tested for equality.This method returns the true object and the invoking object are both Comparator objects and use the same ordering

LISTS

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.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

1. List <data-type> list1= new ArrayList();
2. List <data-type> list2 = new LinkedList();
3. List <data-type> list3 = new Vector();
4. 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.

ARRAYLIST

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.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

ArrayList class declaration

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

```
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable
```

Constructors of Java ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection<? extends E> c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

```java
package javaapplicationarraylist;
import java.util.*;
public class JavaApplicationArrayList {

    public static void main(String[] args) {
      System.out.println("\n\t\t Program for implementation Array List");
      ArrayList obj= new ArrayList();
      System.out.println("\n Inserting some elements in array");
      obj.add(10);
      obj.add(20);
      obj.add(30);
      obj.add(40);
      obj.add(50);
      System.out.println("The array elements are:" +obj);
      System.out.println("\n Inserting some elements in the array in between ");
      obj.add(4,45);
      System.out.println("The array elements are:" +obj);
      System.out.println("\n removing some elements in from the array");
      obj.remove(1);
      System.out.println("The array elements are:" +obj);
      System.out.println("The array elements are:" +obj.size());
    }

}
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

LINKEDLIST

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.
- Java LinkedList class can be used as a list, stack or queue.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

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

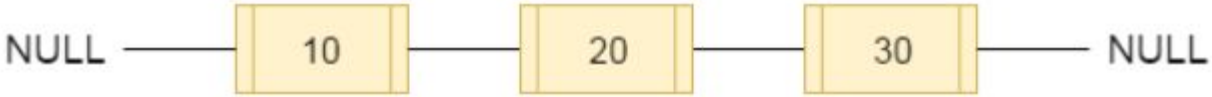
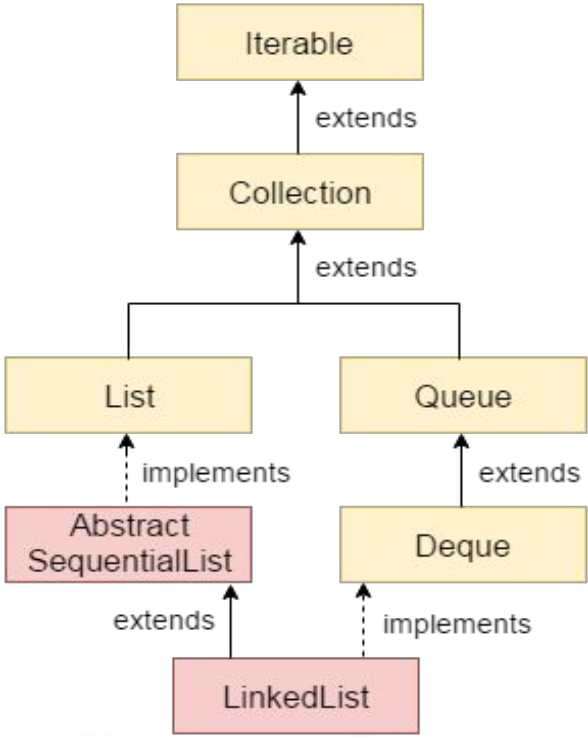


fig- doubly linked list



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

```java
package javaapplicationlinkedlist;
import java.util.*;

public class JavaApplicationLinkedList {

    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());
      }

 }
 }
```

- JavaApplicationLinkedList (run) ×

```
run:
Ravi
Vijay
Ravi
Ajay
BUILD SUCCESSFUL (total time: 0 seconds)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

```
public static void main(String[] args) {
   LinkedList<String> ll=new LinkedList<String>();
        System.out.println("Initial list of elements: "+ll);
        ll.add("Ravi");
        ll.add("Vijay");
        ll.add("Ajay");
        System.out.println("After invoking add(E e) method: "+ll);
        //Adding an element at the specific position
        ll.add(1, "Gaurav");
        System.out.println("After invoking add(int index, E element) method: "+ll);
        LinkedList<String> ll2=new LinkedList<String>();
        ll2.add("Sonoo");
        ll2.add("Hanumat");
        //Adding second list elements to the first list
        ll.addAll(ll2);
        System.out.println("After invoking addAll(Collection<? extends E> c) method: "+ll);
        LinkedList<String> ll3=new LinkedList<String>();
        ll3.add("John");
        ll3.add("Rahul");
        //Adding second list elements to the first list at specific position
        ll.addAll(1, ll3);
        System.out.println("After invoking addAll(int index, Collection<? extends E> c) method: "+ll);
        //Adding an element at the first position
        ll.addFirst("Lokesh");
        System.out.println("After invoking addFirst(E e) method: "+ll);
        //Adding an element at the last position
        ll.addLast("Harsh");
        System.out.println("After invoking addLast(E e) method: "+ll);
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

```
Initial list of elements: []
After invoking add(E e) method: [Ravi, Vijay, Ajay]
After invoking add(int index, E element) method: [Ravi, Gaurav, Vijay, Ajay]
After invoking addAll(Collection<? extends E> c) method: [Ravi, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addAll(int index, Collection<? extends E> c) method: [Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addFirst(E e) method: [Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addLast(E e) method: [Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat, Harsh]
BUILD SUCCESSFUL (total time: 0 seconds)
```

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.

Static Methods for List and Collections

Collection class contains static methods to perform common operations in collection and a list.

Static Methods for List

Method	Description
Void sort(List list)	Sort the given list
void sort(List list,Comparator c)	Sorts the given list with the Comparator
Int BinarySearch(List list,Object key)	Searches the key in the given list using binary search
void reverse(List list)	Reverse the given list.
void shuffle(List list)	Shuffle the specified list randomly
void copy(List dest,List src)	Copies the source list to a destination list.
Void fill(List list,Object obj)	Fills the list with object.

Static Methods for Collection

Method	Description
Object max(Collection c)	This method returns the max object in collection.
Object max(Collection c,Comparator cm)	This method returns the max object in collection using Comparator.
Object min(Collection c)	This method returns the min object in collection.
Object min(Collection c,Comparator cm)	This method returns the minobject in collection using Comparator.
boolean disjoint(Collection c1,Collection c2)	This function returns true if c1 and c2 have no element in common.
int frequency(Collection c,Object o)	It returns number of occurrences of specificied element in the collection

VECTOR

Java Vector class comes under the java.util package. The vector class implements a growable array of objects. Like an array, it contains the component that can be accessed using an integer index.

Vector is very useful if we don't know the size of an array in advance or we need one that can change the size over the lifetime of a program.

Vector implements a dynamic array that means it can grow or shrink as required. It is similar to the ArrayList, but with two differences-

- Vector is synchronized.
- The vector contains many legacy methods that are not the part of a collections framework

Java Vector Class Declaration

1. **public class** Vector<E>
2. **extends** Object<E>
3. **implements** List<E>, Cloneable, Serializable

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Methods used in Vector:

1. **void addElement(Object element):** It inserts the element at the end of the Vector.
2. **int capacity():** This method returns the current capacity of the vector.
3. **int size():** It returns the current size of the vector.
4. **void setSize(int size):** It changes the existing size with the specified size.
5. **boolean contains(Object element):** This method checks whether the specified element is present in the Vector. If the element is been found it returns true else false.
6. **boolean containsAll(Collection c):** It returns true if all the elements of collection c are present in the Vector.
7. **Object elementAt(int index):** It returns the element present at the specified location in Vector.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Methods used in vector

1. **Object first Element():** It is used for getting the first element of the vector.
2. **Object lastElement():** Returns the last element of the array.
3. **Object get(int index):** Returns the element at the specified index.
4. **boolean isEmpty():** This method returns true if Vector doesn't have any element.
5. **boolean removeElement(Object element):** Removes the specified element from vector.
6. **boolean removeAll(Collection c):** It Removes all those elements from vector which are present in the Collection c.
7. **void setElementAt(Object element, int index):** It updates the element of specified index with the given element.

```
package javaapplicationvector;
import java.util.*;
import java.io.*;
public class JavaApplicationVector {

    public static void main(String[] args) throws IOException {
        Vector obj= new Vector(3);
        int ival;
        double dval;
        char ans='y';
        System.out.println("The capacity of vector is:"+obj.capacity());
        System.out.println("The total number of element are"+obj.size());
        System.out.println("\n Inserting the integer:");
        do
        {
            System.out.println("\n enter some integer value");
            ival=getInt();
            obj.addElement(ival);
            System.out.println("\n Do you want to enter more?");
            ans=getChar();
        }
        while(ans=='y');
        System.out.println("The element in the vector are:"+obj);
        System.out.println("\t Inserting the double value:");
        do
        {
           System.out.println("\n Enter some double value");
           dval = getDouble();
           obj.addElement(dval);
           System.out.println("\n Do u want to enter more?");
        }
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

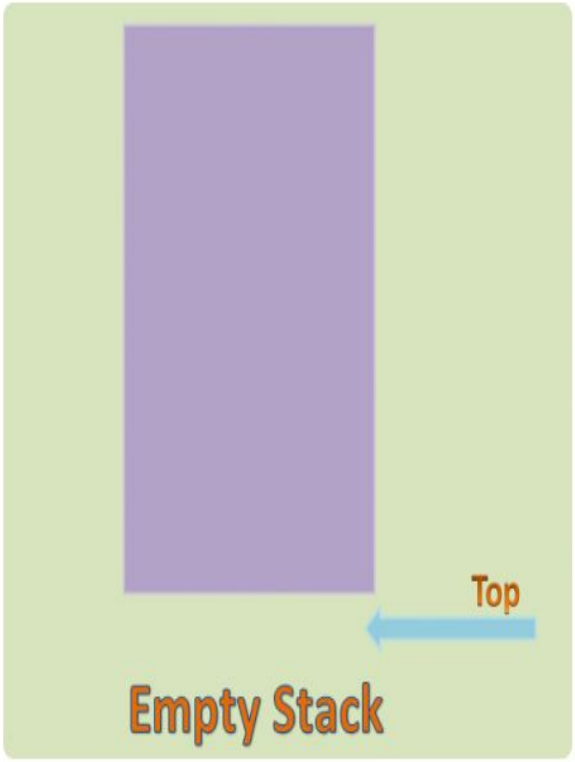
```
    }
   while(ans=='y');
    System.out.println("the elements in the vector are:"+obj);
    System.out.println("\t the size of the vector is:"+obj.size());
    System.out.println("\t The first element in the vector is:"+(Integer)obj.firstElement());
    System.out.println("\t The Last element in the vector is:"+(Integer)obj.lastElement());
    System.out.println("Removing 2 elemnets ");
    obj.remove(0);
    obj.remove(2);
    System.out.println("Now the elements in the vector are"+obj);
   }

public static String getString() throws IOException
{
    InputStreamReader input = new  InputStreamReader(System.in);
    BufferReader b = new BufferReader(input);
    String str = b.readLine();
    return str;
}

public static char getChar() throws IOException
{
    String str = getString();
    return str.charAt(0);
}

private static double getDouble() throws IOException {

    String str=getString();
    return Double.parseDouble(str);

}
```

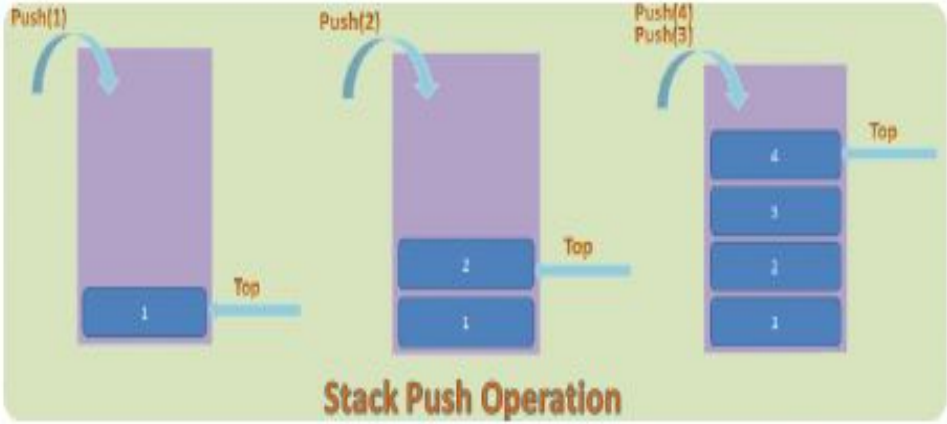
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

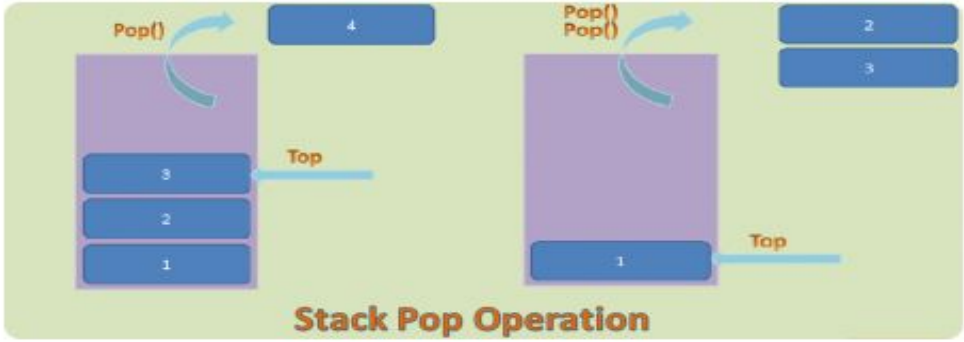
STACK

Java Stack is LIFO object. It extends Vector class but supports only five operations. Java Stack class has only one constructor which is empty or default constructor. So, when we create a Stack, initially it contains no items that mean Stack is empty.

Stack internally has a pointer: TOP, which refers to the top of the Stack element. If Stack is empty, TOP refers to the before first element location. If Stack is not empty, TOP refers to the top element.





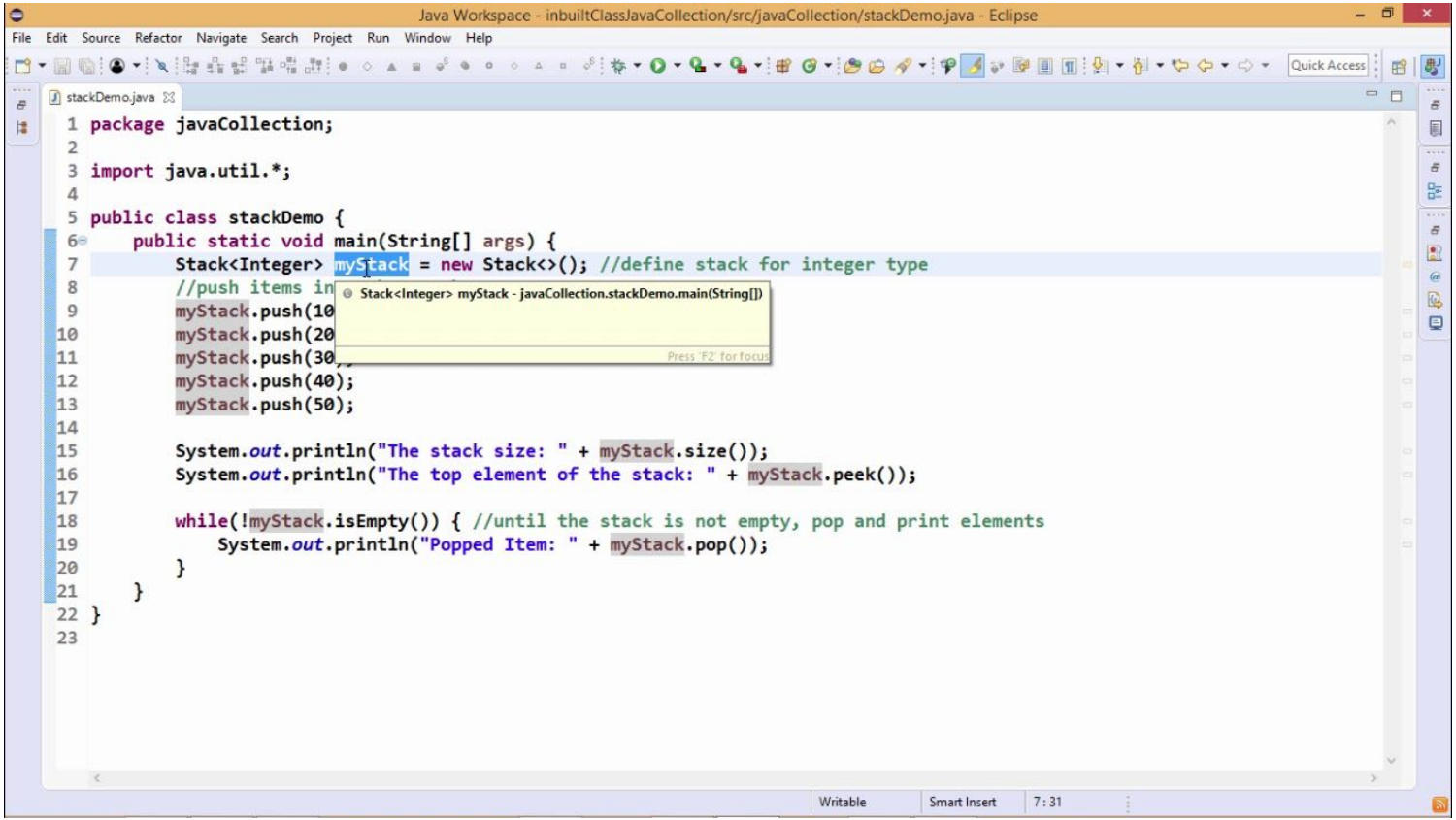


AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Methods used in Stack

Method	Description
boolean empty()	If the stack is empty then it returns true ow it returns false
Object peek()	The element present at the top of the stack is simply displayed
object push(object ele)	This method is useful for pushing the element onto the stack
object pop()	This method is remove the element present at the top of the stack
int search(object ele)	The desired element can be searched using this method.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

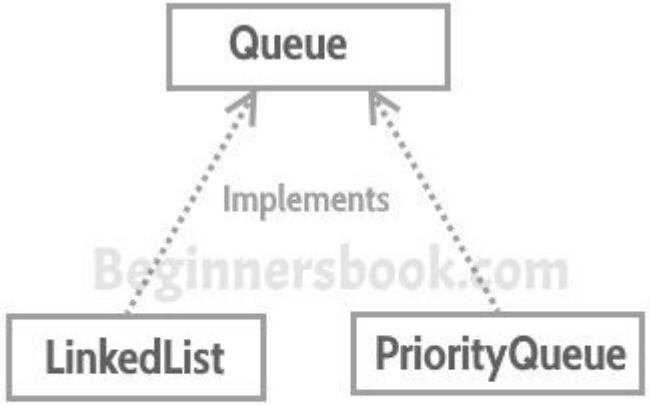


AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

QUEUES

A **Queue** is designed in such a way so that the elements added to it are placed at the end of Queue and removed from the beginning of Queue. The concept here is similar to the queue we see in our daily life,

for example, when a new iPhone launches we stand in a queue outside the apple store, whoever is added to the queue has to stand at the end of it and persons are served on the basis of FIFO (First In First Out), The one who gets the iPhone is removed from the beginning of the queue.



LinkedList & PriorityQueue Classes implements Queue Interface.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Methods used in Queues

1. **boolean add(E e)**: This method adds the specified element at the end of Queue. Returns true if the the element is added successfully or false if the element is not added that basically happens when the Queue is at its max capacity and cannot take any more elements.
2. **E element()**: This method returns the head (the first element) of the Queue.
3. **boolean offer(object)**: This is same as add() method.
4. **E remove()**: This method removes the head(first element) of the Queue and returns its value.
5. **E poll()**: This method is almost same as remove() method. The only difference between poll() and remove() is that poll() method returns null if the Queue is empty.
6. **E peek()**: This method is almost same as element() method. The only difference between peek() and element() is that peek() method returns null if the Queue is empty.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

```java
package javaapplicationqueue;
import java.util.*;
public class JavaApplicationQueue {

    public static void main(String[] args) {

     Queue<Integer> q = new LinkedList<>();

     for(int i=0;i<=5;i++)
         q.add(i+10);
      System.out.println("Elements of Queue:"+q);
      int item = q.remove();
      System.out.println("Removed element: "+item);
      System.out.println("the elements of queue: "+q);
      int head = q.peek();
      System.out.println("The head of the queue is:"+head);
      int size = q.size();
      System.out.println("the size of the queue:"+size);
   }
    }
```

t - JavaApplicationQueue (run) ×

```
run:
Elements of Queue:[10, 11, 12, 13, 14, 15]
Removed element: 10
the elements of queue: [11, 12, 13, 14, 15]
The head of the queue is:11
the size of the queue:5
BUILD SUCCESSFUL (total time: 1 second)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

PRIORITY QUEUES

---

we have seen how a Queue serves the requests based on FIFO(First in First out). Now the question is: **What if we want to serve the request based on the priority rather than FIFO?** In a practical scenario this type of solution would be preferred as it is more dynamic and efficient in nature. This can be done with the help of PriorityQueue, which serves the request based on the priority that we set using Comparator.

```java
package javaapplicationpriorityqueue;

import java.util.PriorityQueue;
public class JavaApplicationPriorityQueue {

    public static void main(String[] args) {
        PriorityQueue<Double> q = new PriorityQueue<Double>();
        System.out.println("\n Inserting element in the queue");
        q.offer(20.2);
        q.offer(10.2);
        q.offer(50.5);
        q.offer(40.4);
        q.offer(30.3);
        q.offer(10.1);
        System.out.println("the size of the queue is:"+q.size());
        System.out.println("\n removing element from the queue");
        while(q.size()>0)
        {
        System.out.println(""+q.peek());
        q.poll();
        }
```

- JavaApplicationPriorityQueue (run) ×

```
 removing element from the queue
10.1
10.2
20.2
30.3
40.4
50.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY