

COLLECTION FRAMEWORK

Collections in Java

- A Collection is a group of individual objects represented as a single unit.
- Java provides Collection Framework which defines several classes and interfaces to represent a group of objects as a single unit.

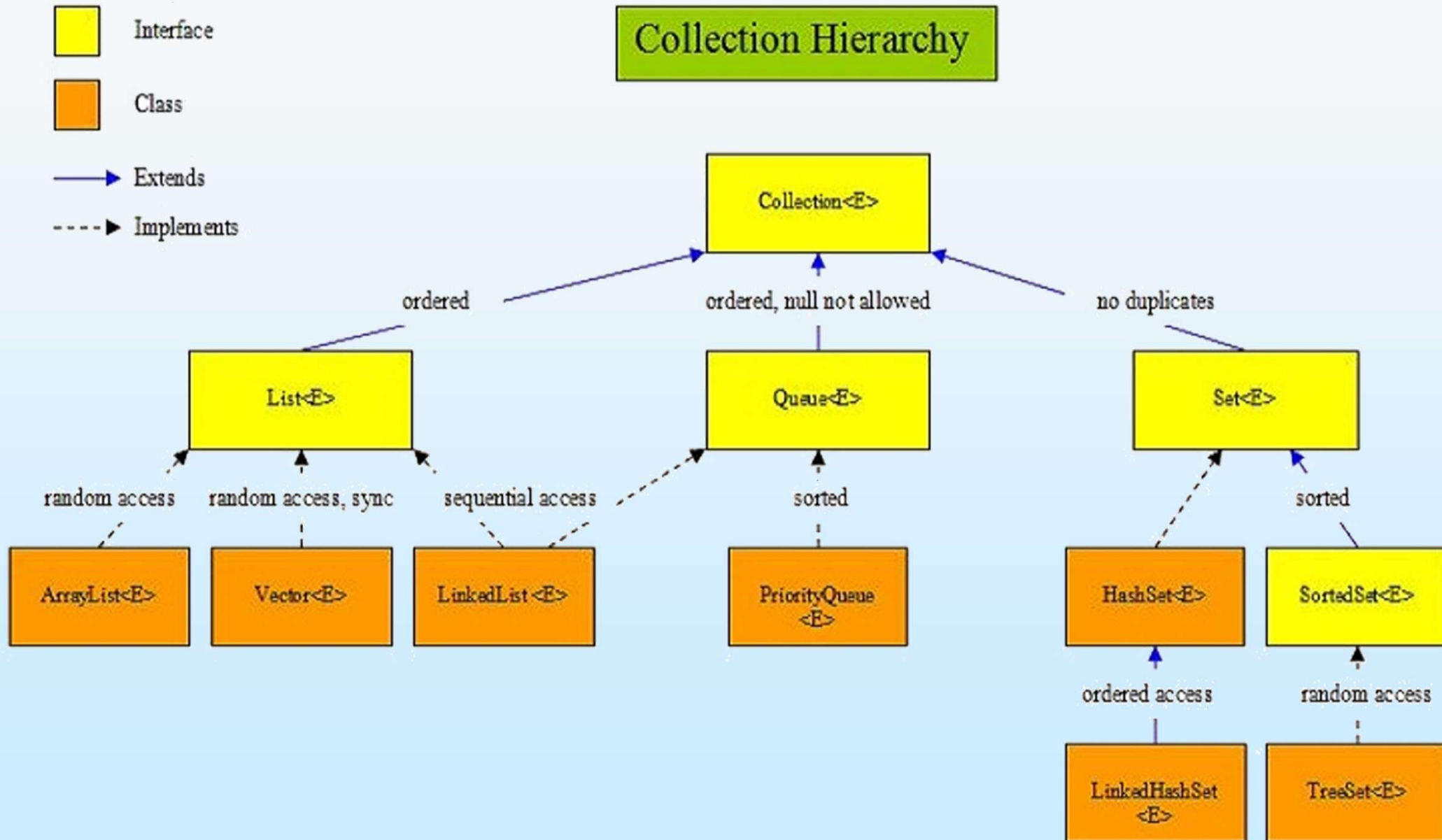
Basics of Collection Framework

□ Features

- Defined in `java.util` package
- Handle `groups` of objects
- Implementation of fundamental collections (dynamic arrays, linked lists etc.) highly efficient (need not `code` them)
- Algorithms (predefined) are defined as `static methods`
- Availability of `Iterator` interface to access elements within a collection one at a time (using the methods defined by the Iterator)
 - ▶ Use of for-each style *for* loops
- Use of `generics` to explicitly state the type of data being stored and to avoid run-time mismatch errors
- Autoboxing / unboxing facilitates the storing of `primitive types` in collections

Basics of Collection Framework

Collection Hierarchy



Collection Interfaces

□ Interfaces in Collection Framework

Interface	Description
Collection	Work with groups of objects; it is at the top of the Collections hierarchy
List	Handles sequences (list of objects)
Queue	Handles lists in FIFO technique
Set	Handles sets
SortedSet	Handles sorted sets
Deque	Extends Queue; Handles double-ended queue
NavigableSet	Extends SortedSet; Handles retrieval of elements based on closest-match searches

Note: The last two interfaces are added in Java SE 6

Collection Interfaces

□ Collection Interface (Continued ...)

□ Methods of Collection interface

Method		Description
boolean	add (Object obj)	<ul style="list-style-type: none">✓ Adds obj to the invoking collection.✓ Returns true if obj was added to the collection.✓ Returns false if obj is not added (if the collection does not allow duplicates).
boolean	addAll (Collection c)	<ul style="list-style-type: none">✓ Adds all the elements of c to the invoking collection.✓ Returns true if the operation succeeded (i.e., the elements were added). Otherwise, returns false.
boolean	contains (Object obj)	<ul style="list-style-type: none">✓ Returns true if obj is an element of the invoking collection. Otherwise, returns false.
boolean	containsAll (Collection c)	<ul style="list-style-type: none">✓ Returns true if the invoking collection contains all elements of c. Otherwise, returns false.

Collection Interfaces

□ Collection Interface (Continued ...)

□ Methods of Collection interface

Method		Description
boolean	<code>remove (Object obj)</code>	<ul style="list-style-type: none">✓ Removes one instance of <code>obj</code> from the invoking collection.✓ Returns true if the element was removed. Otherwise, returns false.
boolean	<code>removeAll (Collection c)</code>	<ul style="list-style-type: none">✓ Removes all elements of <code>c</code> from the invoking collection.✓ Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.
void	<code>clear ()</code>	<ul style="list-style-type: none">✓ Removes all elements from the invoking collection.
boolean	<code>retainAll (Collection c)</code>	<ul style="list-style-type: none">✓ Removes all elements from the invoking collection except those in <code>c</code>.✓ Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.

Collection Interfaces

□ Collection Interface (Continued ...)

□ Methods of Collection interface

Method		Description
boolean	<code>equals (Object obj)</code>	✓ Returns true if the invoking collection and <code>obj</code> are equal. Otherwise, returns false.
int	<code>size ()</code>	✓ Returns the number of elements held in the invoking collection.
boolean	<code>isEmpty ()</code>	✓ Returns true if the invoking collection is empty. Otherwise, returns false.
Iterator	<code>iterator ()</code>	✓ Returns an iterator for the invoking collection.

Collection Interfaces

□ Collection Interface (Continued ...)

□ Methods of Collection interface

Method		Description
Object[]	<code>toArray ()</code>	✓ Returns an array that contains all the elements stored in the invoking collection.
<T> T[]	<code>toArray (T array[])</code>	✓ Returns an array containing only those elements of the invoking collection whose type matches that of the <code>array</code> specified as argument.

Collection Interfaces

□ List Interface (Continued ...)

□ Methods

Method		Description
void	add (int index, Object obj)	✓ Inserts obj into the invoking list at the specified index .
boolean	addAll (int index, Collection c)	✓ Inserts all elements of c into the invoking list at the specified index . ✓ Returns true if the invoking list changes and returns false otherwise.
Object	get (int index)	✓ Returns the object stored at the specified index within the invoking list.
Object	set (int index, Object obj)	✓ Assigns obj to the location specified by index within the invoking list.

Collection Interfaces

□ List Interface (Continued ...)

□ Methods

Method		Description
int	indexOf (Object obj)	<ul style="list-style-type: none">✓ Returns the index of the first instance of obj in the invoking list.✓ Returns -1 if obj is not an element of the list.
int	lastIndexOf (Object obj)	<ul style="list-style-type: none">✓ Returns the index of the last instance of obj in the invoking list.✓ Returns -1 if obj is not an element of the list.
Object	remove (int index)	<ul style="list-style-type: none">✓ Removes the element at position index from the invoking list and returns the deleted element.
Object	remove (Object obj)	<ul style="list-style-type: none">✓ Removes the specified element obj from the invoking list and returns the deleted element.

Collection Interfaces

□ List Interface (Continued ...)

□ Methods

Method		Description
List	<code>subList (int start, int end)</code>	✓ Returns a list that includes elements from <code>start</code> to <code>end</code> -1 in the invoking list.
ListIterator	<code>listIterator ()</code>	✓ Returns an iterator to the start of the invoking list.
ListIterator	<code>listIterator (int index)</code>	✓ Returns an iterator to the invoking list that begins at the specified <code>index</code> .

Collection Classes

□ ArrayList Class

- Supports dynamic arrays (Variable-length array) that grow and shrink as needed
- Can be created with an initial size; when the size exceeded, it is automatically enlarged

Collection Classes

□ ArrayList Class (Continued ...)

□ Constructors:

Constructor	Description
<code>ArrayList ()</code>	✓ Creates an empty array list.
<code>ArrayList (Collection c)</code>	✓ Creates an array list that is initialized with the elements of collection <code>c</code> .
<code>ArrayList (int capacity)</code>	✓ Creates an array list that has the specified initial <code>capacity</code> . ✓ The capacity grows automatically as elements are added to an array list.

Note: The objects in the Collection parameter are of any type.

Collection Classes

ArrayList Class (Continued ...)

Methods:

Method		Description
void	<code>add (int index, Object element)</code>	<ul style="list-style-type: none">✓ Inserts the specified element at the specified index in the list.✓ Throws <code>IndexOutOfBoundsException</code> if index is out of range (index < 0 or index >= size()).
boolean	<code>add (Object obj)</code>	<ul style="list-style-type: none">✓ Appends the specified element obj to the end of the list.

ArrayList Demo-1

```
5 public static void main(String args[])
6 {
7     ArrayList<String> al = new ArrayList<String>();
8
9     System.out.println("Initial size of al: " +
10         al.size());
11
12     al.add("C"); // Add elements to the array list.
13     al.add("A");
14     al.add("E");
15     al.add("B");
16     al.add("D");
17     al.add("F");
18     al.add(1, "A2");
19
20     System.out.println("Size of al: " + al.size());
21
22     System.out.println("Contents of al: " + al);
23
24     al.remove(2);
25
26     System.out.println("Size of al : " + al.size());
27     System.out.println("Contents of al: " + al);
28 }
```


Initial size of a1: 0

Size of a1: 7

Contents of a1: [C, A2, A, E, B, D, F]

Size of a1 : 6

Contents of a1: [C, A2, E, B, D, F]

ArrayList: for-each loop

ArrayList Demo-2

```
6 public static void main(String args[])
7 {
8     ArrayList<Integer> vals = new ArrayList<Integer>();
9
10    vals.add(1);
11    vals.add(2);
12    vals.add(3);
13    vals.add(4);
14    vals.add(5);
15
16    // Use for loop to display the values.
17    System.out.print("Original contents of vals: ");
18    for(int v : vals)
19        System.out.print(v + " ");
20    System.out.println();
21
22    // Now, sum the values by using a for loop.
23    int sum = 0;
24    for(int v : vals)
25        sum += v;
26
27    System.out.println("Sum of values: " + sum);
28 }
```

```
Original contents of vals: 1 2 3 4 5  
Sum of values: 15
```


// Convert an ArrayList into an array.

ArrayList Demo-3

```
6 public static void main(String args[])
7 {
8     ArrayList<Integer> al = new ArrayList<Integer>();
9
10    al.add(1);    al.add(2);    al.add(3);    al.add(4);
11
12    System.out.println("Contents of al: " + al);
13
14    Object arr[] = al.toArray( );
15
16    System.out.println("Contents of arr: " );
17
18    for( Object i : arr )
19        System.out.println( i );
20
21    int sum = 0;
22
23    for( Object i : arr )
24        sum += (int)i;
25
26    System.out.println("Sum is: " + sum);
27 }
```

```
Contents of a1: [11, 22, 33, 44]
```

```
Contents of arr:
```

```
11
```

```
22
```

```
33
```

```
44
```

```
Sum is: 110
```

Collection Classes

ArrayList Class (Continued ...)

Methods:

Method		Description
boolean	<code>addAll (Collection c)</code>	<ul style="list-style-type: none">✓ Appends all the elements in the specified collection <code>c</code> to the end of the list, in the order that they are returned by the specified collection's iterator.✓ Throws <code>NullPointerException</code> if the specified collection <code>c</code> is null.
boolean	<code>addAll (int index, Collection c)</code>	<ul style="list-style-type: none">✓ Inserts all of the elements in the specified collection <code>c</code> into the list, starting at the specified <code>index</code>.✓ Throws <code>NullPointerException</code> if the specified collection <code>c</code> is null.

Collection Classes

ArrayList Class (Continued ...)

Methods:

Method		Description
Object	<code>remove (int index)</code>	<ul style="list-style-type: none">✓ Removes the element at the specified <code>index</code> in the list.✓ Throws <code>IndexOutOfBoundsException</code> if <code>index</code> out of range (<code>index < 0</code> or <code>index >= size()</code>).
void	<code>removeRange (int fromIndex, int toIndex)</code>	<ul style="list-style-type: none">✓ Removes all those elements from the list whose index is between <code>fromIndex</code> (inclusive) and <code>toIndex</code> (exclusive).
void	<code>clear ()</code>	<ul style="list-style-type: none">✓ Removes all elements from the list.
boolean	<code>contains (Object obj)</code>	<ul style="list-style-type: none">✓ Returns true if the list contains the specified element <code>obj</code>.

Collection Classes

ArrayList Class (Continued ...)

Methods:

Method		Description
Object	<code>get (int index)</code>	<ul style="list-style-type: none">✓ Returns the element at the specified <code>index</code> in the list.✓ Throws <code>IndexOutOfBoundsException</code> if <code>index</code> is out of range (<code>index < 0</code> or <code>index >= size()</code>).
Object	<code>set (int index, Object element)</code>	<ul style="list-style-type: none">✓ Replaces the element at the specified <code>index</code> in the list with the specified <code>element</code>.✓ Throws <code>IndexOutOfBoundsException</code> if <code>index</code> is out of range (<code>index < 0</code> or <code>index >= size()</code>).

Collection Classes

ArrayList Class (Continued ...)

Methods:

Method		Description
int	<code>indexOf (Object obj)</code>	<ul style="list-style-type: none">✓ Returns the index of the first occurrence of the specified element <code>obj</code>.✓ Returns -1 if the list does not contain this element.
int	<code>lastIndexOf (Object obj)</code>	<ul style="list-style-type: none">✓ Returns the index of the last occurrence of the specified element <code>obj</code>.✓ Returns -1 if the list does not contain this element.

Collection Classes

ArrayList Class (Continued ...)

Methods:

Method		Description
int	size ()	✓ Returns the number of elements in the list.
void	ensureCapacity (int minCapacity)	✓ Increases the capacity of the ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minCapacity argument.
void	trimToSize ()	✓ Trims the capacity of this ArrayList instance to be the list's current size.
Object []	toArray ()	✓ Returns an array containing all of the elements in the list in the same order. ✓ Throws NullPointerException if the specified array is null.
T[]	toArray(T[] arr)	✓ Converts a list into an array arr[] and returns same.

ArrayList Demo-4

```
5 public static void main(String args[])
6 {
7
8     ArrayList<Double> AL1 = new ArrayList<Double>();
9
10    AL1.add( 11.1 );
11    AL1.add( 12.2 );
12
13    ArrayList<Double> AL2 = new ArrayList<Double>( AL1 );
14
15    AL2.add( 13.3 );
16
17    ArrayList<Double> AL3 = new ArrayList<Double>();
18
19    AL3.addAll( AL2 );
20
21    System.out.println("Size of al : " + AL2.size());
22
23    for( double ele: AL3 )
24        System.out.println( ele );
25 }
```

```
Size of a1 : 3
```

```
11.1
```

```
12.2
```

```
13.3
```

```
3  class Student
4  {
5      int rno;
6      String name;
7      float avg;
8
9      Student() {}
10
11     Student( int r, String n, float a)
12     {
13         rno = r;
14         name = n;
15         avg = a;
16     }
17     void show()
18     {
19         System.out.println(rno+" "+name+" "+avg) ;
20     }
21
22 }
```

```
23 class ArrayListObj
24 {
25     public static void main(String args[])
26     {
27         ArrayList<Student> AL_Stud = new ArrayList<Student>();
28
29         Student s = new Student( 100, "Anil" , 55 );
30
31         AL_Stud.add( s );
32
33         AL_Stud.add( new Student(101,"Vinod",66) );
34
35         AL_Stud.add( new Student(102,"Sachin",88) );
36
37         System.out.println("Contents of AL_Stud: ");
38
39         for( Student e : AL_Stud )
40             e.show();
41     }
42 }
```

Contents of AL_Stud:

100 Anil 55.0

101 Vinod 66.0

102 Sachin 88.0

Iterator and ListIterator

□ Iterator

- An object used to cycle through the elements in a collection
- Implements either the Iterator or the ListIterator interface
 - ▶ ListIterator extends Iterator (to allow bidirectional traversal of a list)
- Declarations

`interface Iterator<E>`

`interface ListIterator<E>`

E specifies the type of objects being iterated

Iterator and ListIterator

□ Iterator class

□ Methods defined

Method		Description
boolean	<code>hasNext ()</code>	<ul style="list-style-type: none">✓ Returns true if there are more elements.✓ Otherwise, returns false.
Object	<code>next ()</code>	<ul style="list-style-type: none">✓ Returns the next element.✓ Throws NoSuchElementException if there is no next element.
void	<code>remove ()</code>	<ul style="list-style-type: none">✓ Removes the current element.

```
1 // Demonstrate iterators.
2 import java.util.*;
3
4 class IteratorDemo
5 {
6     public static void main(String args[])
7     {
8         ArrayList<String> al = new ArrayList<String>();
9
10        al.add("C");
11        al.add("A");
12        al.add("E");
13        al.add("B");
14        al.add("D");
15        al.add("F");
16
17        // Use iterator to display contents of al.
18        System.out.print("Original contents of al: ");
19        Iterator<String> itr = al.iterator();
20        while(itr.hasNext())
21        {
22            String element = itr.next();
23            System.out.print(element + " ");
24        }
25        System.out.println();
```

```
27      // Modify objects being iterated.
28      ListIterator<String> litr = al.listIterator();
29      while(litr.hasNext())
30      {
31          String element = litr.next();
32          litr.set(element + "+");
33      }
34
35      System.out.print("Modified contents of al: ");
36      itr = al.iterator();
37      while(itr.hasNext())
38      {
39          String element = itr.next();
40          System.out.print(element + " ");
41      }
42      System.out.println();
43
44      System.out.print("Modified list backwards: ");
45      while(litr.hasPrevious())
46      {
47          String element = litr.previous();
48          System.out.print(element + " ");
49      }
50  }
51 }
```

Original contents of a1: C A E B D F

Modified contents of a1: C+ A+ E+ B+ D+ F+

Modified list backwards: F+ D+ B+ E+ A+ C+

Iterator and ListIterator

□ ListIterator class

□ Methods defined

Method		Description
boolean	<code>hasNext ()</code>	<ul style="list-style-type: none">✓ Returns true if there are more elements.✓ Otherwise, returns false.
Object	<code>next ()</code>	<ul style="list-style-type: none">✓ Returns the next element.✓ Throws NoSuchElementException if there is no next element.
int	<code>nextIndex ()</code>	<ul style="list-style-type: none">✓ Returns the index of the next element.✓ If there is not a next element, returns the size of the list.

Iterator and ListIterator

□ ListIterator class

□ Methods defined

Method		Description
boolean	<code>hasPrevious ()</code>	<ul style="list-style-type: none">✓ Returns true if there is a previous element.✓ Otherwise, returns false.
Object	<code>previous ()</code>	<ul style="list-style-type: none">✓ Returns the previous element.✓ NoSuchElementException is thrown if there is no previous element.
int	<code>previousIndex ()</code>	<ul style="list-style-type: none">✓ Returns the index of the previous element.✓ If there is not a previous element, returns -1.

Iterator and ListIterator

□ ListIterator class (Continued ...)

□ Methods defined

Method		Description
void	<code>add (Object obj)</code>	✓ Inserts <code>obj</code> into the list in front of the element that will be returned by the next call to <code>next()</code> .
void	<code>remove ()</code>	✓ Removes the current element.
void	<code>set (Object obj)</code>	✓ Assigns <code>obj</code> to the current element. ✓ This is the element last returned by a call to either <code>next()</code> or <code>previous()</code> .


```
import java.util.*;

class IteratorDemo {
    public static void main(String args[]) {
        // Create an array list.
        ArrayList<String> al = new ArrayList<String>();

        // Add elements to the array list.
        al.add("C");
        al.add("A");
        al.add("E");
        al.add("B");
        al.add("D");
        al.add("F");

        // Use iterator to display contents of al.
        System.out.print("Original contents of al: ");
        Iterator<String> itr = al.iterator();
        while(itr.hasNext()) {
            String element = itr.next();
            System.out.print(element + " ");
        }
        System.out.println();

        // Modify objects being iterated.
        ListIterator<String> litr = al.listIterator();
        while(litr.hasNext()) {
            String element = litr.next();
            litr.set(element + "+");
        }
    }
}
```

```
System.out.print("Modified contents of al: ");
    itr = al.iterator();
    while(itr.hasNext()) {
        String element = itr.next();
        System.out.print(element + " ");
    }
    System.out.println();

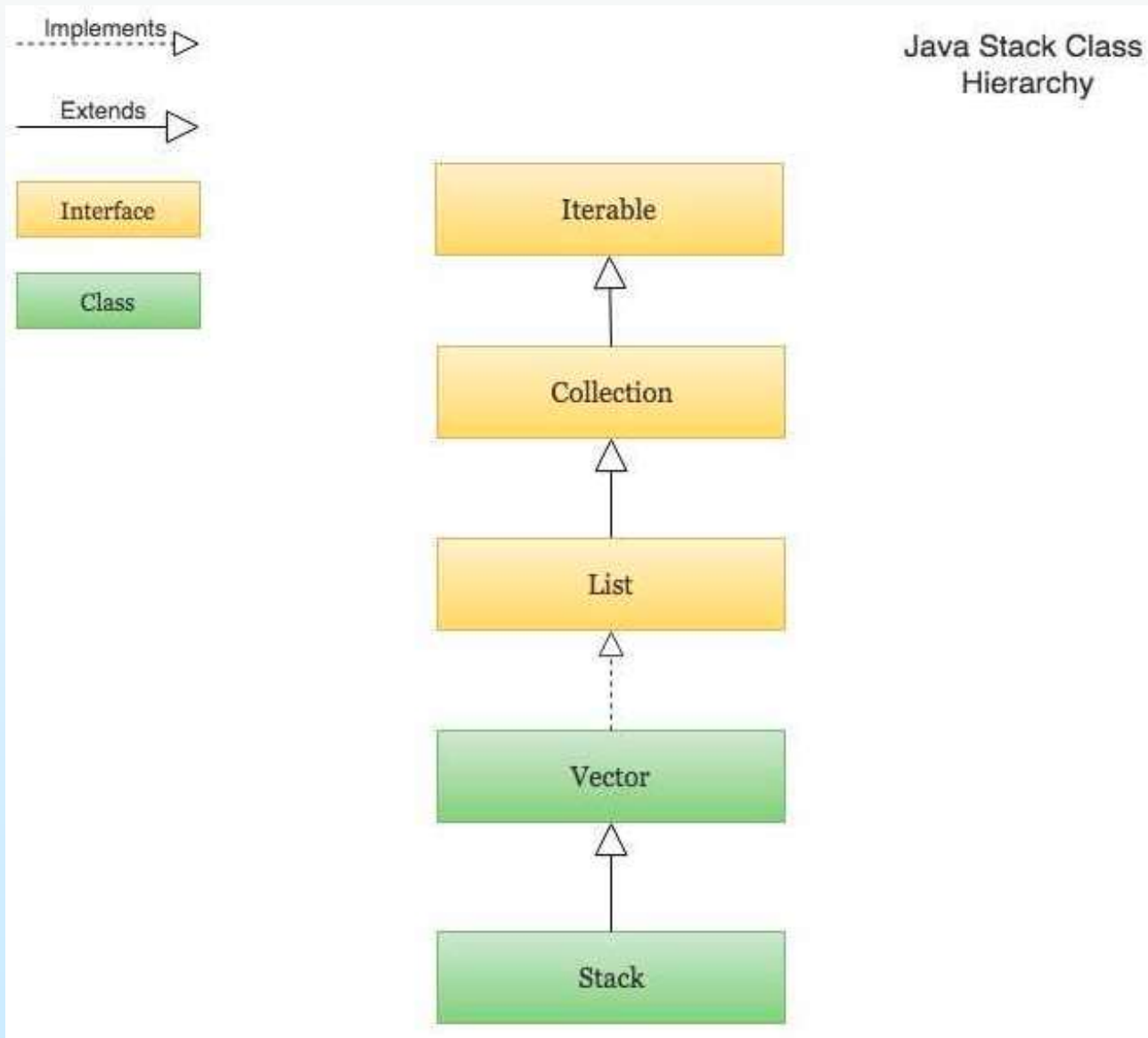
    // Now, display the list backwards.
    System.out.print("Modified list backwards: ");
    while(litr.hasPrevious()) {
        String element = litr.previous();
        System.out.print(element + " ");
    }
    System.out.println();
}
}
```

Iterator and ListIterator

□ Using an Iterator

- All collection classes provide the *iterator()* method that returns an iterator to the start of the collection
- Steps:
 1. Obtain an iterator to the start of the collection by calling the collection's *iterator()* method
 2. Set up a loop that makes a call to *hasNext()*; Iterate through the loop as long as *hasNext()* returns true
 3. Within the loop, obtain each element by calling *next()*
- For collections that implement List, we can obtain an iterator by calling *listIterator()* (E.g., for array list *al*)
 - ▶ `listIterator litr = al.listIterator()` `// Start from the beginning`
 - ▶ `listIterator litr = al.listIterator(al.size())` `// Start from the end`

vector



Collection Classes

□ Vector Class

- Used to create **dynamic array of objects of any type and any number**
- Size need not be specified in advance; can be resized dynamically
- Cannot store simple data types – convert to objects (using wrapper classes)
- Constructors:

Constructor	Description
Vector ()	✓ Empty vector, initial size 0, capacity 10.
Vector (int capacity)	✓ Specify initial capacity.
Vector (int capacity, int increment)	✓ Initial capacity and increment in capacity.
Vector (Collection c)	✓ Creates a vector that contains the elements of Collection c

ArrayList is **not synchronized**. But , Vector is **synchronized**.

Collection Classes

□ Vector Class (Continued ...)

□ Vector methods:

Method		Description
boolean	add(Object ele)	✓ Appends the specified element to the end of this Vector.
void	add (int index, Object element)	✓ Insert element at position index.
void	addElement (Object obj)	✓ Adds obj at the end.
void	insertElementAt (Object obj, int index)	✓ Inserts obj at position index.
boolean	remove (Object obj)	✓ Removes the first occurrence of obj from the vector. ✓ Returns true if removed; false otherwise.
void	removeElementAt (int index)	✓ Removes the element at position index.
void	clear ()	✓ Removes all the elements.

To demonstrate various Vector operations

```
6 public static void main(String args[])
7 {
8     Vector<Integer> v = new Vector<Integer>();
9
10    System.out.println("Initial size: " + v.size());
11    System.out.println("Initial capacity: " +
12                        v.capacity());
13
14    v.addElement(1);
15    v.addElement(2);
16    v.addElement(3);
17    v.addElement(4);
18    // Use an iterator to display contents.
19    Iterator<Integer> vItr = v.iterator();
20
21    System.out.println("\nElements in vector:");
22    while(vItr.hasNext())
23        System.out.print(vItr.next() + " ");
24    System.out.println();
25    // Use an enhanced for loop to display contents.
26    System.out.println("\nElements in vector:");
27    for(int i : v)
28        System.out.print(i + " ");
29    System.out.println();
30 }
```



```
Initial size: 0
```

```
Initial capacity: 10
```

```
Elements in vector:
```

```
1 2 3 4
```

```
Elements in vector:
```

```
1 2 3 4
```

Collection Classes

□ Vector Class (Continued ...)

□ Vector methods:

Method		Description
boolean	contains (Object element)	✓ Returns true if element is found in the vector.
Object	elementAt (int index)	✓ Returns the element at position index .
Object	firstElement ()	✓ Returns the first element (index 0).
Object	lastElement ()	✓ Returns the last element.

Collection Classes

□ Vector Class (Continued ...)

□ Vector methods:

Method		Description
int	<code>indexOf (Object element)</code>	<ul style="list-style-type: none">✓ Returns the index of the first occurrence of <code>element</code>.✓ Returns -1 if not found.
int	<code>indexOf (Object element, int index)</code>	<ul style="list-style-type: none">✓ Returns the index of the first occurrence of <code>element</code> searching forwards from <code>index</code>.✓ Returns -1 if not found.
int	<code>lastIndexOf (Object element)</code>	<ul style="list-style-type: none">✓ Returns the index of the last occurrence of <code>element</code>.✓ Returns -1 if not found.
int	<code>lastIndexOf (Object element, int index)</code>	<ul style="list-style-type: none">✓ Returns the index of the first occurrence of <code>element</code> searching backwards from <code>index</code>.✓ Returns -1 if not found.

Collection Classes

□ Vector Class (Continued ...)

□ Vector methods:

Method		Description
int	capacity ()	✓ Returns the current capacity.
int	size ()	✓ Returns the number of elements.
boolean	isEmpty ()	✓ Returns true if there are no elements in the vector; otherwise, returns false.
void	set (index index, Object element)	✓ Replaces the element at position index with element.
void	copyInto (Object[] anArray)	✓ Copies elements of the vector into the array anArray.

```
6 public static void main(String args[])
7 {
8     Vector<Integer> v1 = new Vector<Integer>();
9
10    v1.addElement(1); v1.addElement(2); v1.addElement(3);
11
12    ArrayList<Integer> AL = new ArrayList<Integer>();
13
14    AL.add( 10 ); AL.add( 20 ); AL.add( 30 ); AL.add( 40 );
15
16    Vector<Integer> v2 = new Vector<Integer>(AL);
17    v2.addElement(55); v2.addAll(v1);
18
19    ListIterator<Integer> litr = v2.listIterator();
20    while(litr.hasNext())
21    {
22        int element = litr.next();
23        litr.set(element + 5 );
24    }
25    while( litr.hasPrevious() )
26    {
27        int element = litr.previous();
28        System.out.println(element + " ");
29    }
30 }
```

8

7

6

60

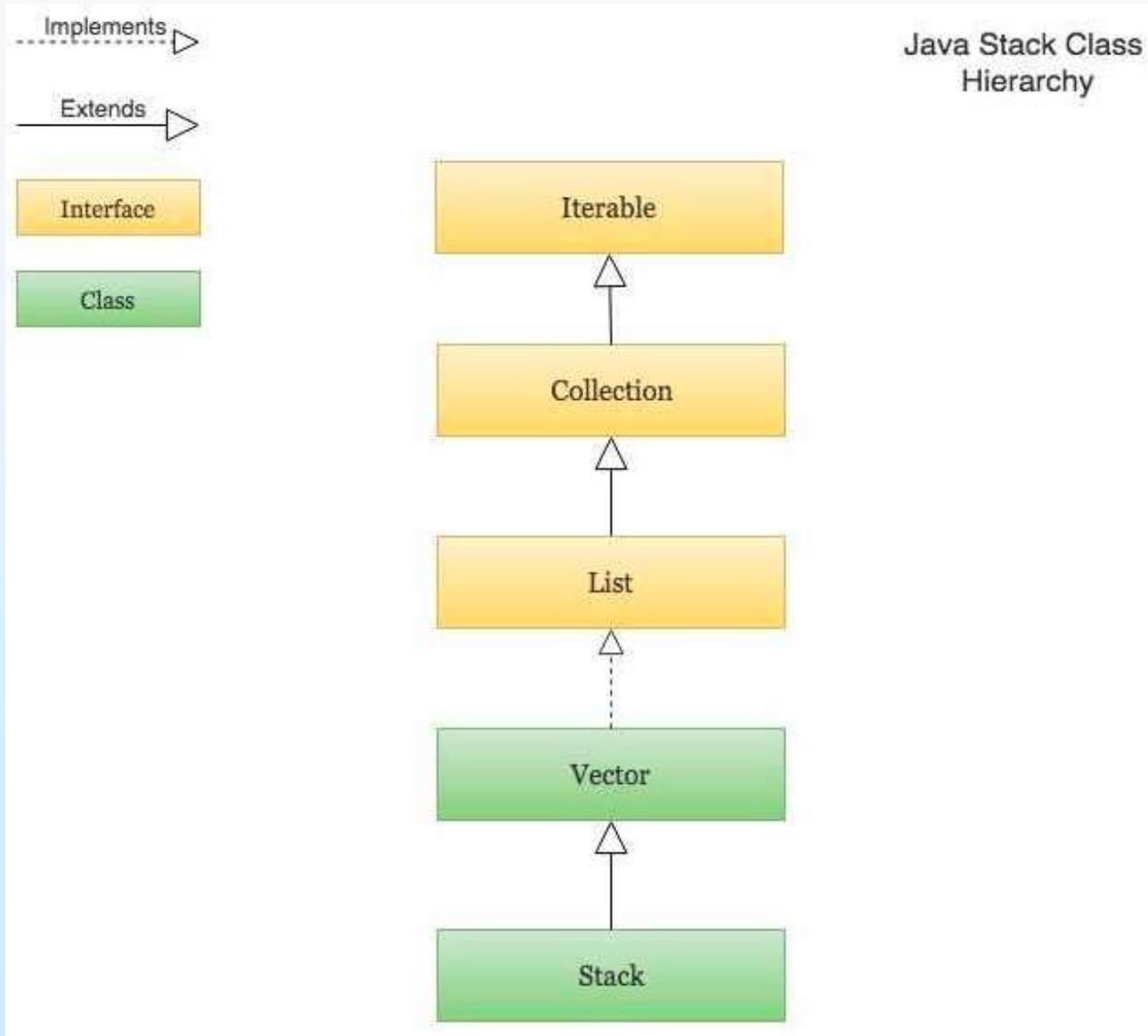
45

35

25

15

stack



Stack class

```
1  // Demonstrate the Stack class.
2  import java.util.*;
3
4  class StackDemo
5  {
6      static void showpush(Stack<Integer> st, int a)
7      {
8          st.push(a);
9          System.out.println("push(" + a + ")");
10         System.out.println("stack: " + st);
11     }
12
13     static void showpop(Stack<Integer> st)
14     {
15         System.out.print("pop -> ");
16         Integer a = st.pop();
17         System.out.println(a);
18         System.out.println("stack: " + st);
19     }
```

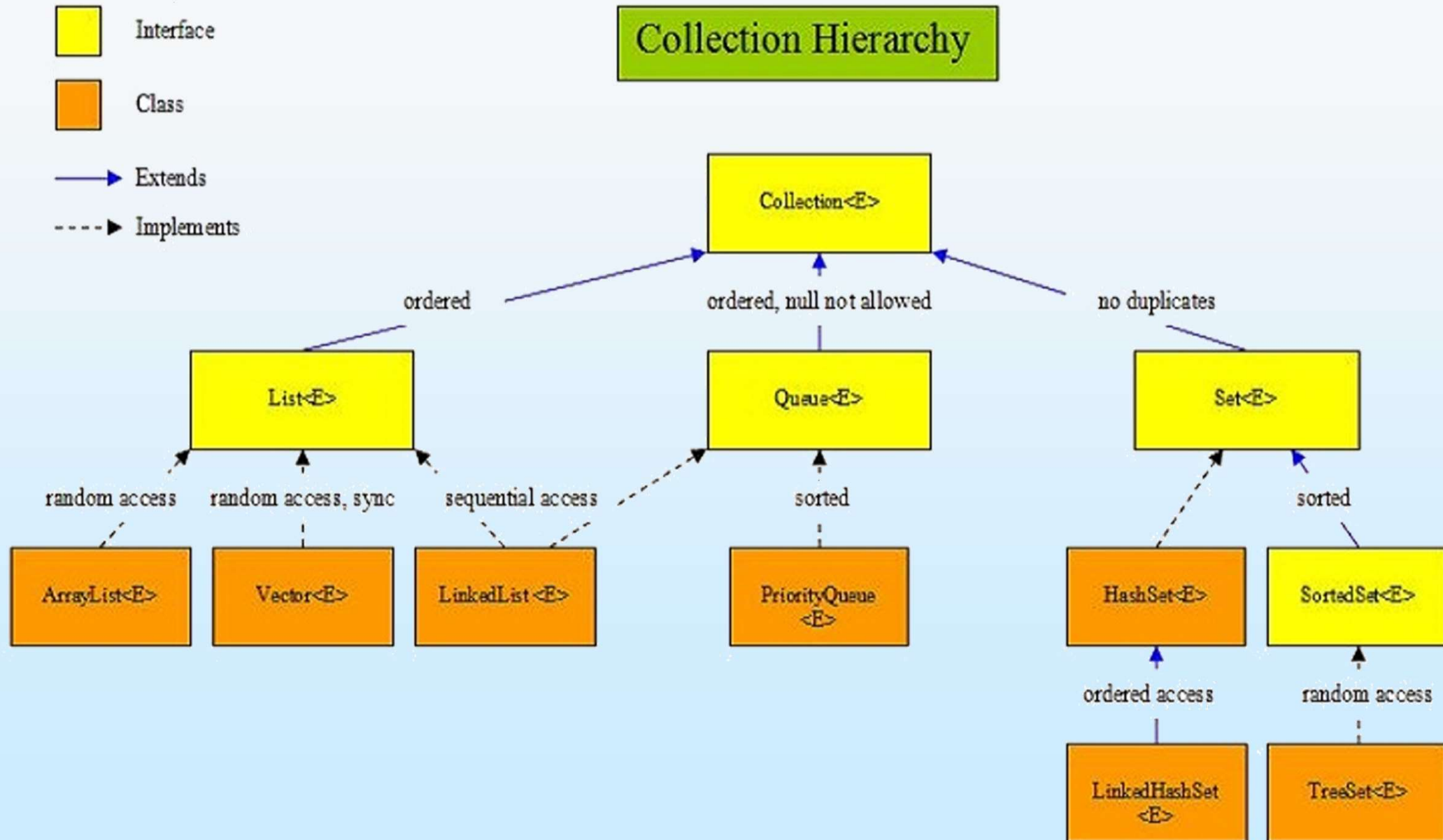


```
21 public static void main(String args[])
22 {
23     Stack<Integer> st = new Stack<Integer>();
24
25     System.out.println("stack: " + st);
26     showpush(st, 42); showpush(st, 66); showpush(st, 99);
27
28     showpop(st); showpop(st); showpop(st);
29
30     try
31     {
32         showpop(st);
33     }
34     catch (EmptyStackException e)
35     {
36         System.out.println("empty stack");
37     }
38 }
39 }
```

```
stack: []  
push(42)  
stack: [42]  
push(66)  
stack: [42, 66]  
push(99)  
stack: [42, 66, 99]  
pop -> 99  
stack: [42, 66]  
pop -> 66  
stack: [42]  
pop -> 42  
stack: []  
pop -> empty stack
```


Basics of Collection Framework

Collection Hierarchy



Collection Classes

❑ LinkedList Class

- ❑ Provides a linked-list data structure
- ❑ Constructors

Constructor	Description
<code>LinkedList ()</code>	✓ Creates an empty linked list.
<code>LinkedList (Collection c)</code>	✓ Creates a linked list that is initialized with the elements of collection <code>c</code> .

- ❑ LinkedList implements List interface
 - ▶ We can use the methods defined by List

Queue

add()- adds elements at the tail of queue

peek()- used to view the head of queue without removing it. It returns Null if the queue is empty.

remove()- Removes and returns the head of the queue. It throws *NoSuchElementException* when the queue is empty.

size()- Returns the number of elements in the queue.

Collection methods as algorithms

The collection framework has the following methods as algorithms.

Method	Description
<code>void sort(List list)</code>	Sorts the elements of the list as determined by their natural ordering.
<code>void sort(List list, Comparator comp)</code>	Sorts the elements of the list as determined by Comparator comp.
<code>void reverse(List list)</code>	Reverses all the elements sequence in list.
<code>void rotate(List list, int n)</code>	Rotates list by n places to the right. To rotate left, use a negative value for n.
<code>void shuffle(List list)</code>	Shuffles the elements in list.
<code>void shuffle(List list, Random r)</code>	Shuffles the elements in the list by using r as a source of random numbers.
<code>void copy(List list1, List list2)</code>	Copies the elements of list2 to list1.
<code>List nCopies(int num, Object obj)</code>	Returns num copies of obj contained in an immutable list. num can not be zero or negative.

// Demonstrate LinkedList.

```
5 public static void main(String[] args) {
6     // Create a linked list.
7     LinkedList<Character> ll = new LinkedList<Character>();
8
9     // Add elements to the linked list.
10    ll.add('A');
11    ll.add('E');
12    ll.add('D');
13    System.out.println("Original contents: " + ll);
14
15    // Demonstrate addLast() and addFirst().
16    ll.addLast('G');
17    ll.addFirst('T');
18    System.out.println("\nAfter calls to addFirst() and addLast().");
19    System.out.println("Contents: " + ll);
20
21    // Add elements at an index.
22    ll.add(2, 'D');
23    ll.add(2, 'C');
24    System.out.println("\nAfter insertions.");
25    System.out.println("Contents: " + ll);
26
27    // Display first and last elements.
28    System.out.println("\nHere are the first and last elements: " +
29        ll.getFirst() + " " + ll.getLast());
```



```
31 // Create a sublist view.
32 List<Character> sub = ll.subList(2, 5);
33 System.out.println("\nContents of sublist view: " + sub);
34
35 // Create a new list that contains the sublist
36 LinkedList<Character> ll2 = new LinkedList<Character>(sub);
37
38 // Remove the elements in ll2 from ll.
39 ll.removeAll(ll2);
40
41 System.out.println("\nAfter removing ll2 from ll.");
42 System.out.println("Contents: " + ll);
43
44 // Remove first and last elements.
45 ll.removeFirst();
46 ll.removeLast();
47
48 System.out.println("\nAfter deleting first and last element: ");
49 System.out.println("Contents: " + ll);
50
51 // Get and set a value through an index.
52 ll.set(0, Character.toLowerCase(ll.get(0)));
53
54 System.out.println("\nAfter change: " + ll);
55 }
```

Original contents: [A, E, D]

After calls to addFirst() and addLast().

Contents: [T, A, E, D, G]

After insertions.

Contents: [T, A, C, D, E, D, G]

Here are the first and last elements: T G

Contents of sublist view: [C, D, E]

After removing 112 from 11.

Contents: [T, A, G]

After deleting first and last element:

Contents: [A]

After change: [a]

Linked list example-2:

```
5  class Address
6  {
7      private String name;
8      private String city;
9      private String state;
10     private String code;
11
12     Address (String n, String c,
13             String st, String cd)
14     {
15         name = n;
16         city = c;
17         state = st;
18         code = cd;
19     }
20
21     public String toString()
22     {
23         return name + "\n" +
24                city + " " + state + " " + code;
25     }
26 }
```

```
28 class MailList
29 {
30     public static void main(String args[])
31     {
32         LinkedList<Address> ml = new LinkedList<Address>();
33
34         // Add elements to the linked list.
35         ml.add(new Address("M.I.T", "Manipal", "Karnataka", "576104"));
36         ml.add(new Address("MAHE", "Manipal", "Karnataka", "576104"));
37         ml.add(new Address("N.I.T.K", "Mangalore", "Karnataka", "575025"));
38
39         // Display the mailing list.
40         for(Address element : ml)
41             System.out.println(element + "\n");
42     }
43 }
```

M.I.T

Manipal Karnataka 576104

MAHE

Manipal Karnataka 576104

N.I.T.K

Mangalore Karnataka 575025

To demonstrate queue:

```
1  import java.util.*;
2  public class QueueDemo
3  {
4      public static void main(String[] args)
5      {
6          Queue<Integer> q = new LinkedList<>();
7
8          for (int i=0; i<5; i++)
9              q.add(i*10);
10
11         System.out.println("Elements of queue-"+q);
12
13         int removed_ele = q.remove();
14         System.out.println("removed element-" + removed_ele);
15
16         q.add( 111 );
17         System.out.println(q);
18
19         int head = q.peek();
20         System.out.println("head of queue-" + head);
21
22         System.out.println("Size of queue-" + q.size());
23     }
24 }
```

```
Elements of queue-[0, 10, 20, 30, 40]  
removed element-0  
[10, 20, 30, 40, 111]  
head of queue-10  
Size of queue-5
```

```
1  // Demonstrate Arrays
2  import java.util.*;
3
4  class ArraysDemo {
5      public static void main(String args[]) {
6
7          // Allocate and initialize array.
8          int array[] = new int[10];
9          for(int i = 0; i < 10; i++)
10             array[i] = -3 * i;
11
12         // Display, sort, and display the array.
13         System.out.print("Original contents: ");
14         display(array);
15         Arrays.sort(array);
16         System.out.print("Sorted: ");
17         display(array);
18
19         // Fill and display the array.
20         Arrays.fill(array, 2, 6, -1);
21         System.out.print("After fill(): ");
22         display(array);
```



```
24     // Sort and display the array.
25     Arrays.sort(array);
26     System.out.print("After sorting again: ");
27     display(array);
28
29     // Binary search for -9.
30     System.out.print("The value -9 is at location ");
31     int index =
32         Arrays.binarySearch(array, -9);
33
34     System.out.println(index);
35 }
36
37 static void display(int array[]) {
38     for(int i: array)
39         System.out.print(i + " ");
40
41     System.out.println();
42 }
43 }
```

```
Original contents: 0 -3 -6 -9 -12 -15 -18 -21 -24 -27  
Sorted: -27 -24 -21 -18 -15 -12 -9 -6 -3 0  
After fill(): -27 -24 -1 -1 -1 -1 -9 -6 -3 0  
After sorting again: -27 -24 -9 -6 -3 -1 -1 -1 -1 0  
The value -9 is at location 2
```

Demonstrate several algorithms.

```
6 public static void main(String[] args) {
7
8     ArrayList<Integer> AL = new ArrayList<Integer>();
9
10    // Put items in the list.
11    AL.add(5); AL.add(-15); AL.add(20); AL.add(0);
12    AL.add(-2); AL.add(-5); AL.add(12); AL.add(1);
13
14    // Display original list.
15    System.out.print("Original list: ");
16    for(int i : AL)
17        System.out.printf("%d\t",i);
18
19    System.out.println();
20
21    // Sort the list.
22    Collections.sort(AL);
23    System.out.print("List sorted: ");
24    for(int i : AL)
25        System.out.printf("%d\t",i);
26
27    System.out.println("\n");
```

```
29 // Search the list.
30 System.out.println("Using binarySearch() to find X.");
31 int k = Collections.binarySearch(AL, 5);
32 if(k >= 0)
33     System.out.println("X found. Index is " + k);
34
35 // Reverse the list.
36 Collections.reverse(AL);
37 System.out.print("List reversed: ");
38 for(int i : AL)
39     System.out.printf("%d\t", i);
40
41 // Rotate the List.
42 Collections.rotate(AL, 3);
43 System.out.print("List rotated: ");
44 for(int i : AL)
45     System.out.printf("%d\t", i);
46
47 // Replace all -5's with 50
48 Collections.replaceAll(AL, -5, 50);
49 System.out.print("After replacements: ");
50 for(int i : AL)
51     System.out.printf("%d\t", i);
52 }
53 }
```



```
Original list: 5      -15      20      0      -2      -5      12      1
List sorted:  -15      -5      -2      0      1      5      12      20

Using binarySearch() to find X.
X found. Index is 5

List reversed: 20      12      5      1      0      -2      -5      -15
List rotated:  -2      -5      -15      20      12      5      1      0
After replacements: -2      50      -15      20      12      5      1      0
```

Algorithms demo-2:

```
6 public static void main(String[] args) {
7
8     // Create a linked list.
9     LinkedList<Character> ll = new LinkedList<Character>();
10
11    // Put items in the list.
12    for(int i = 0; i < 26; i+=2) {
13        ll.add((char) ('A' + i));
14        ll.add((char) ('Z' - i));
15    }
16
17    // Display original list.
18    System.out.print("Original list: ");
19    for(char ch : ll)
20        System.out.print(ch);
21
22    System.out.println();
23
24    // Sort the list.
25    Collections.sort(ll);
26    System.out.print("List sorted: ");
27    for(char ch : ll)
28        System.out.print(ch);
```

```
32 // Search the list.
33 System.out.println("Using binarySearch() to find X.");
34 int i = Collections.binarySearch(l1, 'X');
35 if(i >= 0)
36     System.out.println("X found. Index is " + i);
37
38 // Reverse the list.
39 Collections.reverse(l1);
40 System.out.print("List reversed: ");
41 for(char ch : l1)
42     System.out.print(ch);
43
44 // Rotate the List.
45 Collections.rotate(l1, 5);
46 System.out.print("List rotated: ");
47 for(char ch : l1)
48     System.out.print(ch);
```

```
50 // Create a new list.
51 ll = new LinkedList<Character>();
52
53 // Add a string to it.
54 String str = "this is a test";
55 for(char ch : str.toCharArray())
56     ll.add(ch);
57
58 System.out.print("Here is the new list: ");
59 for(char ch : ll)
60     System.out.print(ch);
61
62 // Replace all t's with *
63 Collections.replaceAll(ll, 't', '*');
64 System.out.print("After replacements: ");
65 for(char ch : ll)
66     System.out.print(ch);
67 }
68 }
```


Original list: AZCXEVGTIRKPMNOLQJSHUFWDYB

List sorted: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Using `binarySearch()` to find X.

X found. Index is 23

List reversed: ZYXWVUTSRQPONMLKJIHGFEDCBA

List rotated: EDCBAZYXWVUTSRQPONMLKJIHGF

Here is the new list: this is a test

After replacements: *his is a *es*

The End