COLLECTION FRAMEWORK

Collections in Java

- A Collection is a group of individual objects represented as a single unit.

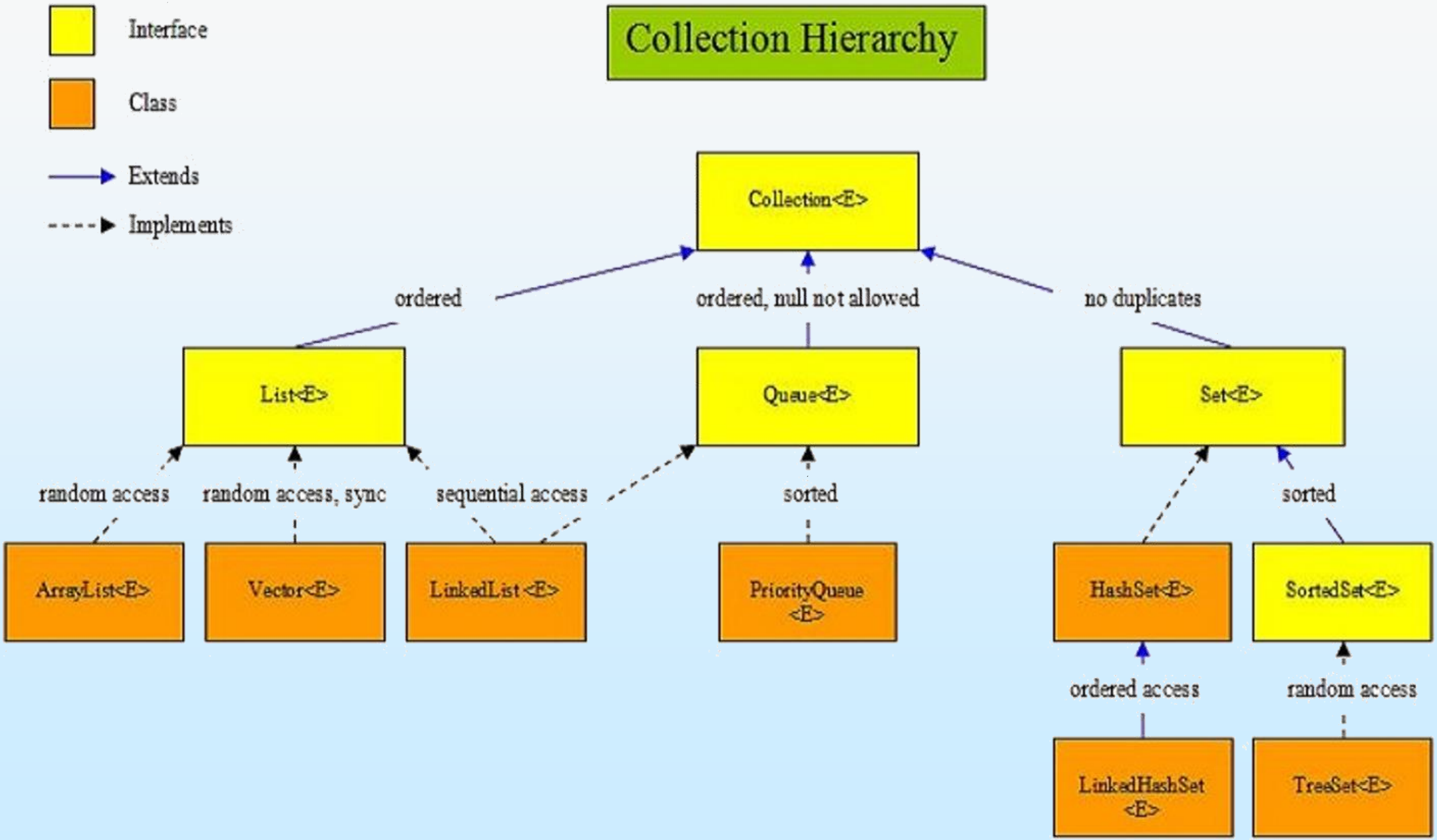
- Java provides Collection Framework which defines several classes and interfaces to represent a group of objects as a single unit.

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

- **Interfaces in Collection Framework** (Continued …)
  - Other interfaces

Interface	Description
Comparator	Compare objects
Iterator	Cycle through a collection (Unidirectional)
ListIterator	Cycle through a collection (Bidirectional)
RandomAccess	Supports random access to elements of the Collection

Collection Interfaces

Collection Interface (Continued …)

- Methods of Collection interface

Method		Description
boolean	add (Object obj)	✓ 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)	✓ 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)	✓ Returns true if obj is an element of the invoking collection. Otherwise, returns false.
boolean	containsAll (Collection c)	✓ 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	remove (Object obj)	✓ Removes one instance of obj from the invoking collection. ✓ Returns true if the element was removed. Otherwise, returns false.
boolean	removeAll (Collection c)	✓ Removes all elements of c from the invoking collection. ✓ Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.
void	clear ( )	✓ Removes all elements from the invoking collection.
boolean	retainAll (Collection c)	✓ Removes all elements from the invoking collection except those in c. ✓ 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	equals (Object obj)	✓ Returns true if the invoking collection and obj are equal. Otherwise, returns false.
int	size ( )	✓ Returns the number of elements held in the invoking collection.
boolean	isEmpty ( )	✓ Returns true if the invoking collection is empty. Otherwise, returns false.
Iterator	iterator ( )	✓ Returns an iterator for the invoking collection.

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)	✓ 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)	✓ 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)	✓ Removes the element at position index from the invoking list and returns the deleted element.
Object	remove (Object obj)	✓ Removes the specified element obj from the invoking list and returns the deleted element.

Collection Interfaces

List Interface (Continued …)

Methods

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

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
ArrayList ( )	✓ Creates an empty array list.
ArrayList (Collection c)	✓ Creates an array list that is initialized with the elements of collection c.
ArrayList (int capacity)	✓ Creates an array list that has the specified initial capacity. ✓ 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	add (int index, Object element)	✓ Inserts the specified element at the specified index in the list. ✓ Throws IndexOutOfBoundsException if index is out of range (index < 0 or index >= size()).
boolean	add (Object obj)	✓ 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 al: 0
Size of al: 7
Contents of al: [C, A2, A, E, B, D, F]
Size of al : 6
Contents of al: [C, A2, E, B, D, F]
```

Arraylist:

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 al: [11, 22, 33, 44]
Contents of arr:
11
22
33
44
Sum is: 110
```

Collection Classes

ArrayList Class (Continued …)

- Methods:

Method		Description
boolean	addAll (Collection c)	✓ Appends all the elements in the specified collection c to the end of the list, in the order that they are returned by the specified collection's iterator. ✓ Throws NullPointerException if the specified collection c is null.
boolean	addAll (int index, Collection c)	✓ Inserts all of the elements in the specified collection c into the list, starting at the specified index. ✓ Throws NullPointerException if the specified collection c is null.

Collection Classes

ArrayList Class (Continued …)

- Methods:

Method		Description
Object	remove (int index)	✓ Removes the element at the specified index in the list. ✓ Throws IndexOutOfBoundsException if index out of range (index < 0 or index >= size()).
void	removeRange (int fromIndex, int toIndex)	✓ Removes all those elements from the list whose index is between fromIndex (inclusive) and toIndex (exclusive).
void	clear ( )	✓ Removes all elements from the list.
boolean	contains (Object obj)	✓ Returns true if the list contains the specified element obj.

Collection Classes

ArrayList Class (Continued …)

- Methods:

Method		Description
Object	get (int index)	✓ Returns the element at the specified index in the list. ✓ Throws IndexOutOfBoundsException if index is out of range (index < 0 or index >= size()).
Object	set (int index, Object element)	✓ Replaces the element at the specified index in the list with the specified element. ✓ Throws IndexOutOfBoundsException if index is out of range (index < 0 or index >= size()).

Collection Classes

ArrayList Class (Continued …)

- Methods:

Method		Description
int	indexOf (Object obj)	✓ Returns the index of the first occurrence of the specified element obj. ✓ Returns -1 if the list does not contain this element.
int	lastIndexOf (Object obj)	✓ Returns the index of the last occurrence of the specified element obj. ✓ 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 al : 3
11.1
12.2
13.3
```

ArrayList Demo-5

```
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	hasNext ( )	✓ Returns true if there are more elements. ✓ Otherwise, returns false.
Object	next ( )	✓ Returns the next element. ✓ Throws NoSuchElementException if there is no next element.
void	remove ( )	✓ Removes the current element.

```java
// Demonstrate iterators.
import java.util.*;

class IteratorDemo
{
  public static void main(String args[])
  {
    ArrayList<String> al = new ArrayList<String>();

    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();
```

```
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 al: C A E B D F
Modified contents of al: 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	hasNext ( )	✓ Returns true if there are more elements. ✓ Otherwise, returns false.
Object	next ( )	✓ Returns the next element. ✓ Throws NoSuchElementException if there is no next element.
int	nextIndex ( )	✓ 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	hasPrevious ( )	✓ Returns true if there is a previous element. ✓ Otherwise, returns false.
Object	previous ( )	✓ Returns the previous element. ✓ NoSuchElementException is thrown if there is no previous element.
int	previousIndex ( )	✓ 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	add (Object obj)	✓ Inserts obj into the list in front of the element that will be returned by the next call to next( ).
void	remove ( )	✓ Removes the current element.
void	set (Object obj)	✓ Assigns obj to the current element. ✓ This is the element last returned by a call to either next( ) or previous( ).

```
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 along 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

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	indexOf (Object element)	✓ Returns the index of the first occurrence of element. ✓ Returns -1 if not found.
int	indexOf (Object element, int index)	✓ Returns the index of the first occurrence of element searching forwards from index. ✓ Returns -1 if not found.
int	lastIndexOf (Object element)	✓ Returns the index of the last occurrence of element. ✓ Returns -1 if not found.
int	lastIndexOf (Object element, int index)	✓ Returns the index of the first occurrence of element searching backwards from index. ✓ 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

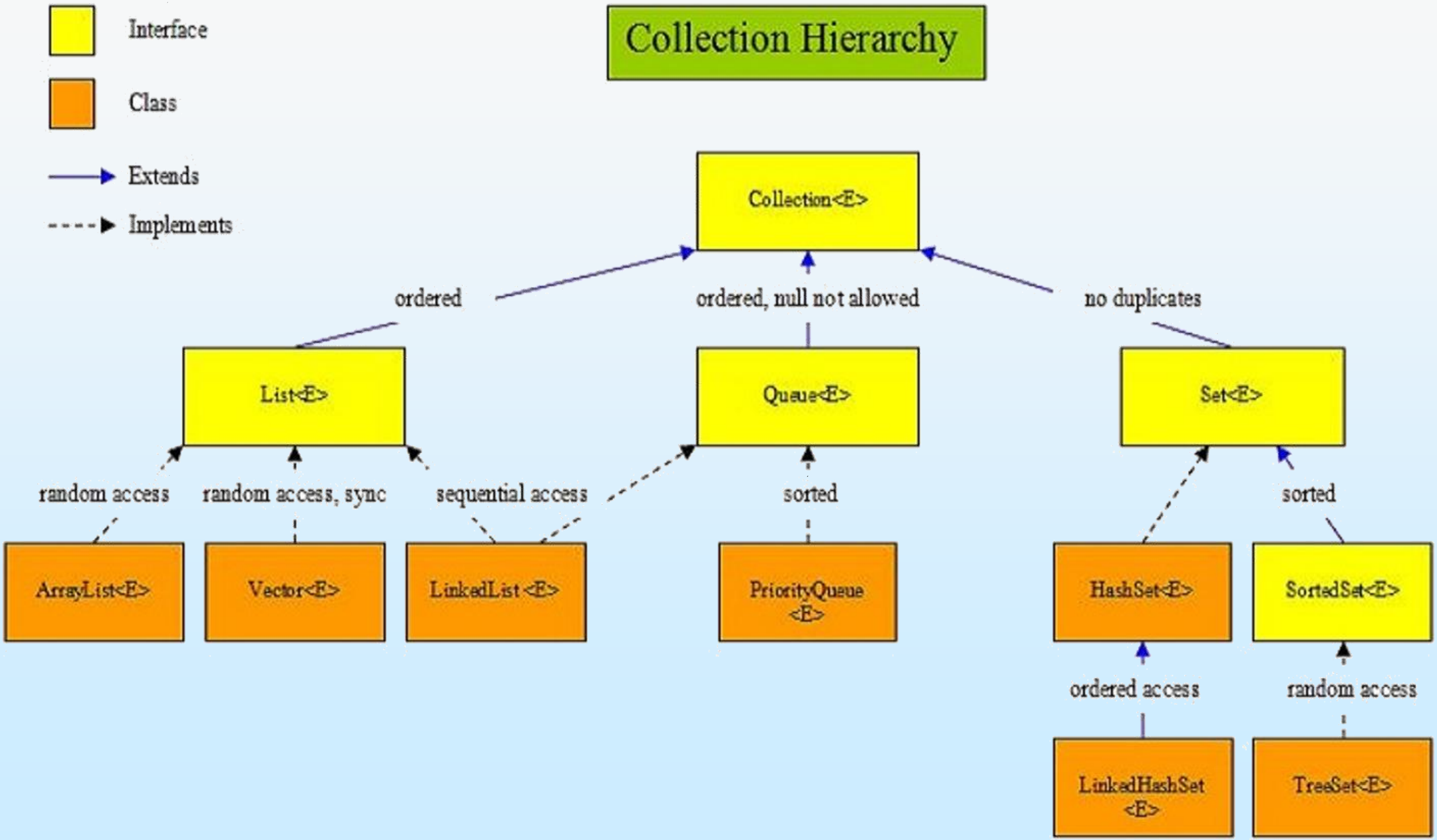
35

25

15

Basics of Collection Framework

Collection Hierarchy

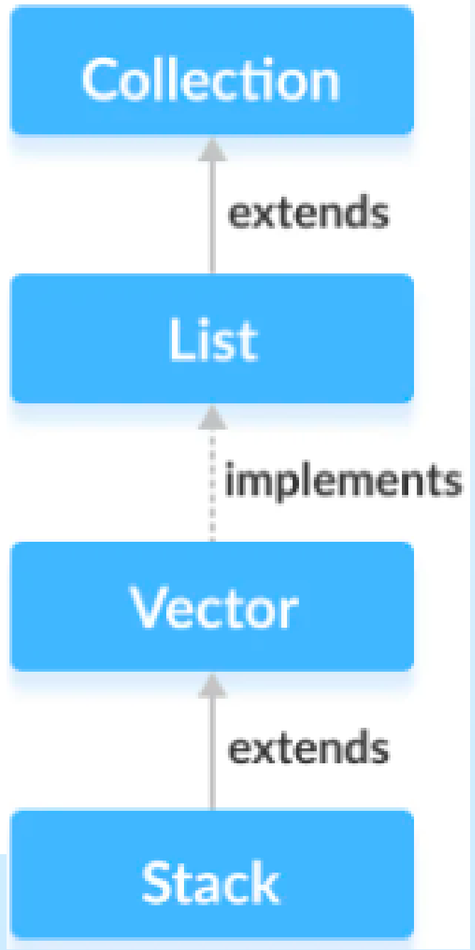


Stack class

```java
// Demonstrate the Stack class.
import java.util.*;

class StackDemo
{
  static void showpush(Stack<Integer> st, int a)
  {
    st.push(a);
    System.out.println("push(" + a + ")");
    System.out.println("stack: " + st);
  }

  static void showpop(Stack<Integer> st)
  {
    System.out.print("pop -> ");
    Integer a = st.pop();
    System.out.println(a);
    System.out.println("stack: " + st);
  }
```

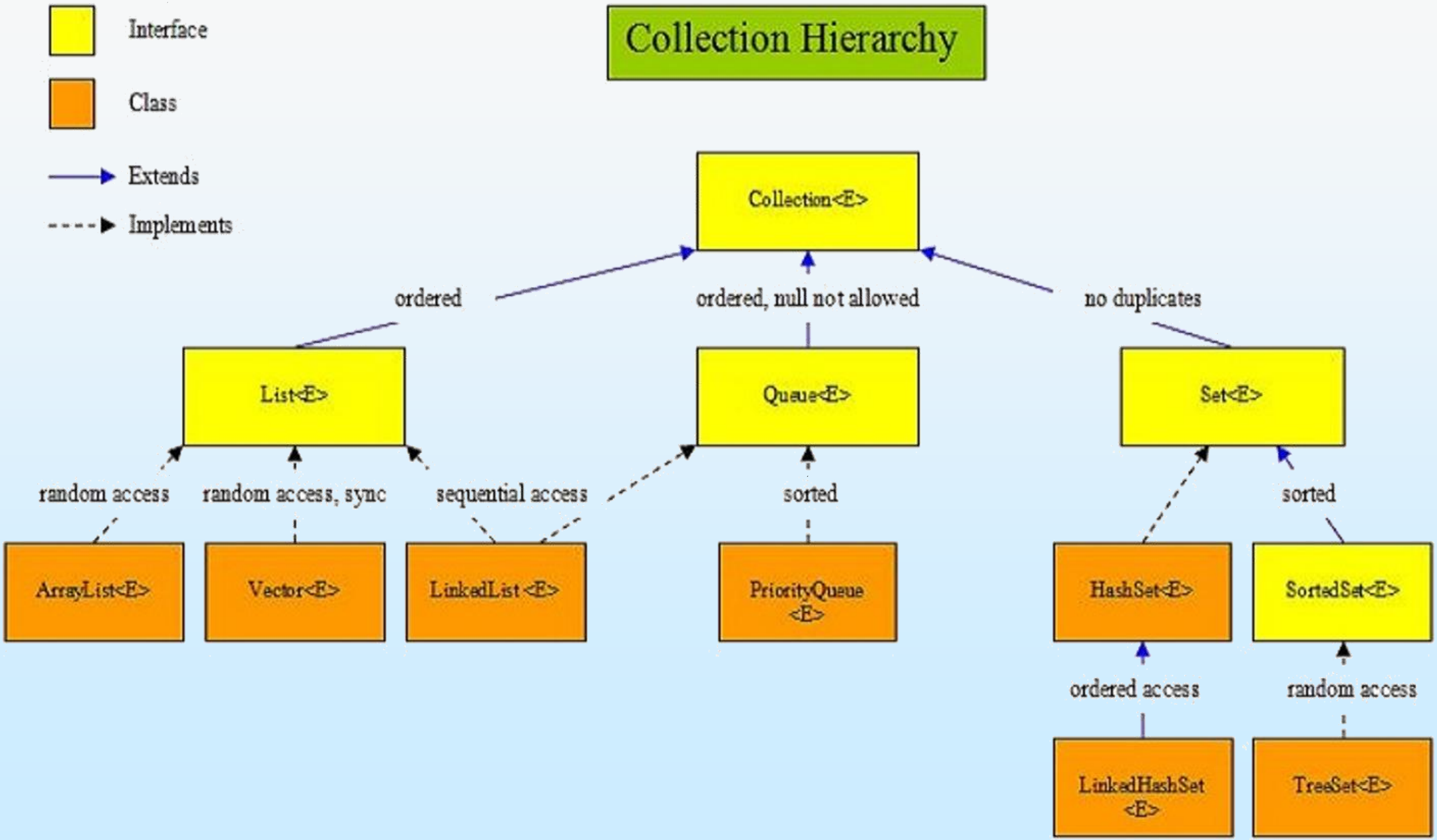


```
21      public static void main(String args[])
22      {
23        Stack<Integer> st = new Stack<Integer>();
24
25        System.out.println("stack: " + st);
26        showpush(st, 42); showpush(st, 66); showpush(st, 99);
27
28        showpop(st); showpop(st); showpop(st);
29
30        try
31        {
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
LinkedList ( )	✓ Creates an empty linked list.
LinkedList (Collection c)	✓ Creates a linked list that is initialized with the elements of collection c.

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

// 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 ll2 from ll.
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  }
```

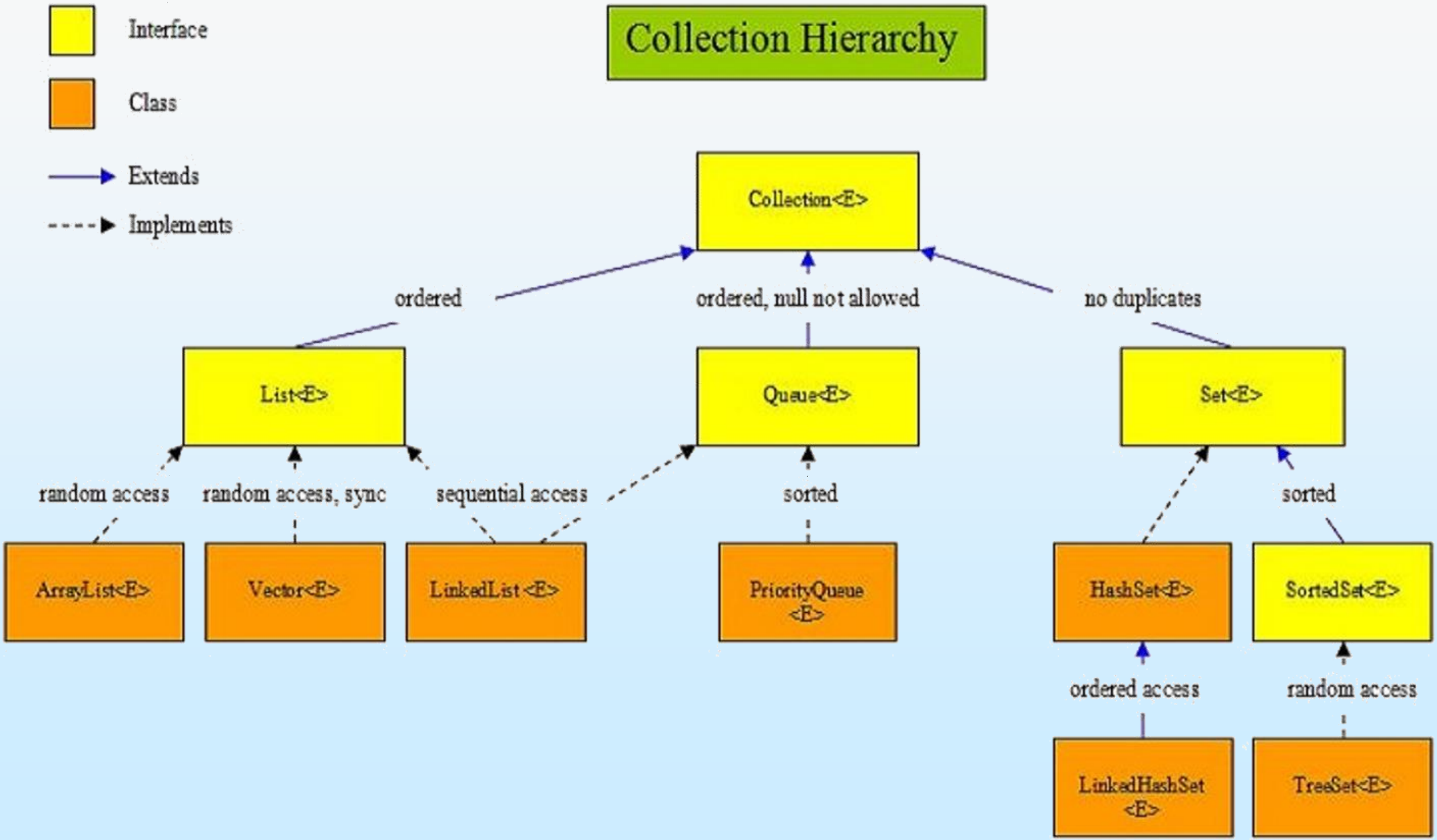
```
M.I.T
Manipal Karnataka 576104

MAHE
Manipal Karnataka 576104

N.I.T.K
Mangalore Karnataka 575025
```

Basics of Collection Framework

Collection Hierarchy



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.

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
```

```java
// Demonstrate Arrays
import java.util.*;

class ArraysDemo {
  public static void main(String args[]) {

    // Allocate and initialize array.
    int array[] = new int[10];
    for(int i = 0; i < 10; i++)
      array[i] = -3 * i;

    // Display, sort, and display the array.
    System.out.print("Original contents: ");
    display(array);
    Arrays.sort(array);
    System.out.print("Sorted: ");
    display(array);

    // Fill and display the array.
    Arrays.fill(array, 2, 6, -1);
    System.out.print("After fill(): ");
    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
```

Collection methods as algorithms

The collection framework has the following methods as algorithms.

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

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(ll, 'X');
35        if(i >= 0)
36          System.out.println("X found. Index is " + i);
37
38        // Reverse the list.
39        Collections.reverse(ll);
40        System.out.print("List reversed: ");
41        for(char ch : ll)
42          System.out.print(ch);
43
44        // Rotate the List.
45        Collections.rotate(ll, 5);
46        System.out.print("List rotated:  ");
47        for(char ch : ll)
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*
```

Demonstrate ArrayDeque. First the use the deque as as a stack. Then, use it as a FIFO queue.

```
 7  class ArrayDequeDemo {
 8    public static void main(String[] args) {
 9      // Create an array deque.
10      ArrayDeque<Character> adq = new ArrayDeque<Character>();
11
12      System.out.println("Using adq as a stack.");
13      // Use adq like a stack.
14      System.out.print("Pushing: ");
15
16      // push items on the stack
17      for(char ch = 'A'; ch <= 'Z'; ch++) {
18        adq.push(ch);
19        System.out.print(ch);
20      }
21
22      System.out.println();
23
24      // now, pop them off
25      System.out.print("Popping: ");
26      while(adq.peek() != null)
27        System.out.print(adq.pop());
28
29      System.out.println("\n");
```

```
31        System.out.println("Using adq as a FIFO queue.");
32        // Now, use adq as a FIFO queue.
33        System.out.print("Queueing: ");
34        for(char ch = 'A'; ch <= 'Z'; ch++) {
35          adq.offerLast(ch);
36          System.out.print(ch);
37        }
38
39        System.out.println();
40
41        // now, remove them
42        System.out.print("Removing: ");
43        while(adq.peek() != null)
44          System.out.print(adq.pollFirst());
45      }
46  }
```

```
Using adq as a stack.
Pushing: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Popping: ZYXWVUTSRQPONMLKJIHGFEDCBA

Using adq as a FIFO queue.
Queueing: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Removing: ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Collection Classes

HashSet Class

- Collection that uses a hash table for storage of elements
- Extends AbstractSet class and implements Set interface, declared as

```
class HashSet <E> extends AbstractSet implements Set
```

- Execution time of *add*(), *contains*(), *remove*() and *size*() remains constant even for large sets

Collection Classes

HashSet Class (Continued …)

Constructors

Constructor	Description
HashSet ( )	✓ Creates an empty set with an initial capacity 16 and default load factor 0.75
HashSet (int initialCapacity)	✓ Creates a set with specified initialCapacity and default load factor 0.75
HashSet (int initialCapacity, float loadFactor)	✓ Creates a set with specified initialCapacity and loadFactor.
HashSet (Collection c)	✓ Creates a set containing elements in the specified collection c.

Collection Classes

- **HashSet Class** (Continued …)
  - HashSet does not define any additional methods than defined by its superclass and interface
  - No guarantee about the order of elements
  - If sorted set is required, TreeSet can be used

Collection Classes

LinkedHashSet Class

- Maintains a linked list of the entries in the set, in the order in which they are inserted
- Extends HashSet, declared as

  class LinkedHashSet <E> extends HashSet

- Does not add any new methods
- Constructors – similar to HashSet

Collection Classes

- **TreeSet Class**
  - Creates a collection that uses a tree for storage
  - Objects are stored in sorted (ascending) order (access is fast)
  - Extends AbstractSet class and implements NavigableSet interface, declared as

```
class TreeSet <E> extends AbstractSet implements NavigableSet
```

Collection Classes

TreeSet Class (Continued …)

Constructors

Constructor	Description
TreeSet ( )	✓ Creates an empty tree set sorted in ascending order according to the natural order of its elements.
TreeSet (Collection c)	✓ Creates a tree set that contains elements of collection c.
TreeSet (SortedSet ss)	✓ Creates a tree set that contains all elements of sorted set ss.
TreeSet (Comparator comp)	✓ Creates an empty tree set that will be sorted according to the comparator specified by comp.

```
Import java.util.*;
class TreeSetDemo {
  public static void main(String args[]) {
    // Create a tree set.
    TreeSet<String> ts = new TreeSet<String>();

    // Add elements to the tree set.
    ts.add("C");
    ts.add("A");
    ts.add("B");
    ts.add("E");
    ts.add("F");
    ts.add("D");

    System.out.println(ts);
  }
}
```

Collection Classes

- **PriorityQueue Class**
  - Creates a queue that is prioritized based on the queue's comparator
  - Extends AbstractQueue class and implements Queue interface
  - Declared as

    class PriorityQueue<E> extends AbstractQueue implements Queue
  - A dynamic data structure

Collection Classes

PriorityQueue Class (Continued …)

Constructors

Constructor	Description
PriorityQueue ( )	✓ Creates a priority queue with default capacity (11).
PriorityQueue (int initialCapacity)	✓ Creates a priority queue with specified initialCapacity.
PriorityQueue (Collection c)	✓ Creates a priority queue containing elements in the specified collection c.

Collection Classes

PriorityQueue Class (Continued …)

Constructors

Constructor	Description
PriorityQueue (int initialCapacity, Comparator comp)	✓ Creates a priority queue with the specified initialCapacity that orders its elements according to the specified comparator comp.
PriorityQueue (PriorityQueue c)	✓ Creates a priority queue containing the elements of the specified priority queue c.
PriorityQueue (SortedSet ss)	✓ Creates a priority queue containing elements in the specified sorted set ss.

Collection Classes

- **ArrayDeque Class**
  - Creates a dynamic array (no capacity restrictions)
  - Extends AbstractCollection class and implements Deque interface
  - Declared as

    class ArrayDeque<E> extends AbstractCollection implements Deque
  - No methods of its own

Collection Classes

ArrayDeque Class (Continued …)

- Constructors

Constructor	Description
ArrayDeque ( )	✓ Creates an empty deque with an initial capacity of 16 elements.
ArrayDeque (int capacity)	✓ Creates a deque with the specified initialCapacity.
ArrayDeque (Collection c)	✓ Creates a deque that is initialized with the elements of collection c.

```
class ArrayDequeDemo {
  public static void main(String args[]) {
    // Create a tree set.
    ArrayDeque<String> adq = new
ArrayDeque<String>();

    // Use an ArrayDeque like a stack.
    adq.push("A");
    adq.push("B");
    adq.push("D");
    adq.push("E");
    adq.push("F");

    System.out.print("Popping the stack:
");

    while(adq.peek() != null)
      System.out.print(adq.pop() + " ");

    System.out.println();
  }
}
```

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	hasNext ( )	✓ Returns true if there are more elements. ✓ Otherwise, returns false.
Object	next ( )	✓ Returns the next element. ✓ Throws NoSuchElementException if there is no next element.
void	remove ( )	✓ Removes the current element. ✓ Throws IllegalStateException if an attempt is made to call remove() that is not preceded by a call to next().

Iterator and ListIterator

ListIterator class

- Methods defined

Method		Description
boolean	hasNext ( )	✓ Returns true if there are more elements. ✓ Otherwise, returns false.
Object	next ( )	✓ Returns the next element. ✓ Throws NoSuchElementException if there is no next element.
int	nextIndex ( )	✓ 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	hasPrevious ( )	✓ Returns true if there is a previous element. ✓ Otherwise, returns false.
Object	previous ( )	✓ Returns the previous element. ✓ NoSuchElementException is thrown if there is no previous element.
int	previousIndex ( )	✓ 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	add (Object obj)	✓ Inserts obj into the list in front of the element that will be returned by the next call to next( ).
void	remove ( )	✓ Removes the current element. ✓ Throws IllegalStateException if an attempt is made to call remove() that is not preceded by a call to next() or previous().
void	set (Object obj)	✓ Assigns obj to the current element. ✓ This is the element last returned by a call to either next( ) or previous( ).

```
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 along 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

Iterator and ListIterator

For-Each alternative

- The for-each version of the *for* loop can be used to cycle through a collection, provided
  - Contents of the collection are not to be modified
  - Elements are not required in reverse order

ForEach Demo

Comparators

Comparator Interface

- A generic interface (to specify the order of sorting) that has the declaration

  interface Comparator <T>

  - T specifies the type of objects being compared

Comparators

Comparator Interface

- Methods:

Method		Description
int	compare (Object obj1, Object obj2)	✓ Returns zero if the objects are equal;, a positive value if obj1 is greater than obj2, a negative value, otherwise. ✓ By overriding compare( ), we can alter the way that objects are ordered. ✓ Throws ClassCastException if objects are not type-compatible

ComparatorDemo 1 ComparatorDemo 2

The End