```java
package weiss.util; // Fig 6.9-10, pg 192-4.

public interface Collection
        extends java.io.Serializable
{
    int size( );
    boolean isEmpty( );
    boolean contains( Object x );
    boolean add( Object x );
    boolean remove( Object x );
    void clear( );
    Iterator iterator( );
    Object [ ] toArray( );
}

public interface Iterator
{
    boolean hasNext( );
    Object next( );
    void remove( );
}
```

```java
// Fig 6.16,6.17, pg 201-2
package weiss.util;

public interface List
        extends Collection
{
    Object get( int idx );
    Object set( int idx,
            Object newVal );
    ListIterator listIterator( int pos );
}

public interface ListIterator
        extends Iterator
{
    boolean hasPrevious( );
    Object previous( );
    void remove( );
}
```

```java
package weiss.util;  // Fig 15.10, pg 501
public abstract class AbstractCollection
            implements Collection
{
    public boolean isEmpty( )  { return size( ) == 0; }
    public void clear( ) {
        Iterator itr = iterator( );
        while( itr.hasNext( ) ) {
            itr.next( );
            itr.remove( );
        }
    }
    public Object [ ] toArray( ) { /* not shown */ }
    public boolean contains( Object x ) {
        if( x == null ) return false;
        Iterator itr = iterator( );
        while( itr.hasNext( ) )
            if( x.equals( itr.next( ) ) ) return true;
        return false;
    }
    public boolean remove( Object x ) {
        if( x == null ) return false;
        Iterator itr = iterator( );
        while( itr.hasNext( ) )
            if( x.equals( itr.next( ) ) ) {
                itr.remove( );
                return true;
            }
        return false;
    }
}
```

```java
package weiss.util;  // Fig 15.12, pg 503
public class ArrayList extends AbstractCollection
                implements List
{
    private static final int DEFAULT_CAPACITY = 10;
    private static final int NOT_FOUND = -1;
    private Object [ ] theItems;
    private int theSize;
    private int modCount = 0;
    public ArrayList( )  { clear( ); }
    public ArrayList( Collection other ) {
        clear( );
        Iterator itr = other.iterator( );
        while( itr.hasNext( ) ) add( itr.next( ) );
    }
    public int size( ) { return theSize; }
    public Object get( int idx )  {
        if( idx < 0 || idx >= size( ) )
            throw new ArrayIndexOutOfBoundsException();
        return theItems[ idx ];
    }
    public Object set( int idx, Object newVal ) {
        if( idx < 0 || idx >= size( ) )
            throw new ArrayIndexOutOfBoundsException();
        Object old = theItems[ idx ];
        theItems[ idx ] = newVal;
        return old;
    }
    public boolean contains( Object x ) {
        return findPos( x ) != NOT_FOUND;
    }

    private int findPos( Object x ) {
        for( int i = 0; i < size( ); i++ )
            if( x == null ) {
                if( theItems[ i ] == null )  return i;
            }
            else if( x.equals( theItems[ i ] ) )  return i;
        return NOT_FOUND;
    }
    public boolean add( Object x )  {
        if( theItems.length == size( ) )  {
            Object [ ] old = theItems;
            theItems = new Object[ theItems.length * 2 + 1 ];
            for( int i = 0; i < size( ); i++ )   theItems[ i ] = old[ i ];
        }
        theItems[ theSize++ ] = x;
        modCount++;
        return true;
    }
    public boolean remove( Object x )  {
        int pos = findPos( x );
        if( pos == NOT_FOUND ) return false;
        else {
            remove( pos );
            return true;
        }
    }
    public Object remove( int idx )  {
        Object removedItem = theItems[ idx ];
        for( int i = idx; i < size( ) - 1; i++ )
            theItems[ i ] = theItems[ i + 1 ];
        theSize--;  modCount++;    return removedItem;
    }
```

3

```java
public Iterator iterator( ) { return new ArrayListIterator( 0 );  }
public ListIterator listIterator( int idx ) {return new ArrayListIterator( idx );}
private class ArrayListIterator implements ListIterator  {
    private int current;
    private int expectedModCount = modCount;
    private boolean nextCompleted = false; private boolean prevCompleted = false;
    ArrayListIterator( int pos )  {
       if( pos < 0 || pos > size( ) ) throw new IndexOutOfBoundsException( );
       current = pos;
    }
    public boolean hasNext( )  {
       if( expectedModCount != modCount )
          throw new ConcurrentModificationException( );
       return current < size( );
    }
    public boolean hasPrevious( ) { /* OMITTED */ }
    public Object next( )  {
       if( !hasNext( ) )  throw new NoSuchElementException( );
       nextCompleted = true;   prevCompleted = false;
       return theItems[ current++ ];
    }
    public Object previous( ) { /* OMITTED */ }
    public void remove( ) {
       if( expectedModCount != modCount )
          throw new ConcurrentModificationException( );
       if( nextCompleted ) ArrayList.this.remove( --current );
       else if( prevCompleted )  ArrayList.this.remove( current );
       else throw new IllegalStateException( );
       prevCompleted = nextCompleted = false;  expectedModCount++;
    }
  }
}
```

```java
package weiss.nonstandard;
class ListNode
{
   public ListNode( Object theElement ) { this( theElement, null ); }
   public ListNode( Object theElement, ListNode n ) {
      element = theElement;
      next   = n;
   }
   public Object   element;
   public ListNode next;
}

public class LinkedListIterator
{
   LinkedListIterator( ListNode theNode ){ current = theNode; }
   public boolean isValid( ) { return current != null; }
   public Object retrieve( )
           {return isValid( ) ? current.element : null;}
   public void advance( ) {
      if( isValid( ) )  current = current.next;
    }
    ListNode current;   // Current position
}
```

```java
package weiss.util;

public class LinkedList extends AbstractCollection
            implements List
{
    public LinkedList( ) { clear( ); }
    public LinkedList( Collection other ) {
        clear( );
        Iterator itr = other.iterator( );
        while( itr.hasNext( ) )
            add( itr.next( ) );
    }
    public int size( ) { return theSize; }
    public boolean contains( Object x ) {
        return findPos( x ) != NOT_FOUND;
    }
    private Node findPos( Object x ) {
        for( Node p = beginMarker.next;
                p != endMarker; p = p.next )
            if( x == null ) {
                if( p.data == null ) return p;
            }
            else if( x.equals( p.data ) ) return p;
        return NOT_FOUND;
    }
    public boolean add( Object x ) {
        addLast( x );
        return true;
    }
    public void addFirst( Object x ) { add( 0, x); }
    public void addLast( Object x ) { add( size( ), x ); }

    public void add( int idx, Object x ) {
        Node p = getNode( idx );
        Node newNode = new Node( x, p.prev, p );
        newNode.prev.next = newNode;
        p.prev = newNode;
        theSize++;
        modCount++;
    }
    public Object getFirst( ) {
        if( isEmpty( ) )
            throw new NoSuchElementException( );
        return getNode( 0 ).data;
    }
    public Object getLast( ) {
        if( isEmpty( ) )
            throw new NoSuchElementException( );
        return getNode( size( ) - 1 ).data;
    }
    public Object get( int idx ) {return getNode( idx ).data;}
    private Node getNode( int idx ) {
        Node p;
        if( idx < 0 || idx > size( ) )
            throw new IndexOutOfBoundsException( );
        if( idx < size( ) / 2 ) {
            p = beginMarker.next;
            for( int i = 0; i < idx; i++ )   p = p.next;
        } else {
            p = endMarker;
            for( int i = size( ); i > idx; i-- )  p = p.prev;
        }
        return p;
    }
}
```

6

```java
public Object removeFirst( ) {
   if( isEmpty( ) ) throw new NoSuchElementException( );
   return remove( getNode( 0 ) );
}
public Object removeLast( ) {
   if( isEmpty( ) ) throw new NoSuchElementException( );
   return remove( getNode( size( ) - 1 ) );
}
public boolean remove( Object x ) {
   Node pos = findPos( x );
   if( pos == NOT_FOUND )   return false;
   else {
      remove( pos );
      return true;
   }
}
public Object remove( int idx ) { return remove( getNode( idx ) );}
private Object remove( Node p ) {
   p.next.prev = p.prev;
   p.prev.next = p.next;
   theSize--;
   modCount++;
   return p.data;
}
public void clear( ) {
   beginMarker = new Node( "BEGINMARKER", null, null );
   endMarker = new Node( "ENDMARKER", beginMarker, null );
   beginMarker.next = endMarker;
   theSize = 0;
   modCount++;
}
```

```java
private class LinkedListIterator implements ListIterator
{
    private Node current;
    private Node lastVisited = null;
    private boolean lastMoveWasPrev = false;
    private int expectedModCount = modCount;

    public LinkedListIterator( int idx ){current = getNode( idx );}
    public boolean hasNext( ) {
        if( expectedModCount != modCount )
            throw new ConcurrentModificationException( );
        return current != endMarker;
    }
    public Object next( ) {
        if( !hasNext( ) ) throw new NoSuchElementException( );
        Object nextItem = current.data;
        lastVisited = current;
        current = current.next;
        lastMoveWasPrev = false;
        return nextItem;
    }
    public void remove( ){
        if( expectedModCount != modCount )
            throw new ConcurrentModificationException( );
        if( lastVisited == null ) throw new IllegalStateException( );
        LinkedList.this.remove( lastVisited );
        lastVisited = null;
        if( lastMoveWasPrev )
            current = current.next;
        expectedModCount++;
    }
```

```java
    public boolean hasPrevious( )
    {
        if( expectedModCount != modCount )
            throw new ConcurrentModificationException( );
        return current != beginMarker.next;
    }

    public Object previous( )
    {
        if( expectedModCount != modCount )
            throw new ConcurrentModificationException( );
        if( !hasPrevious( ) )
            throw new NoSuchElementException( );

        current = current.prev;
        lastVisited = current;
        lastMoveWasPrev = true;
        return current.data;
    }
}
```

Fig 17.30, page 562

```java
public interface Stack
{
    public Object push( Object x );
    public Object pop( );
    public boolean isEmpty( );
}

public interface Queue
{
    public boolean isEmpty( );
    public void enqueue( Object x );
    public Object dequeue( );
}
```

```java
public class BinarySearch // Fig 5.11, pg168
{
  public static final int NOT_FOUND = -1;

  public static int binarySearch
          ( Comparable [ ] a, Comparable x )
  {
    int low = 0;
    int high = a.length - 1;
    int mid;
    while( low <= high )
    {
      mid = ( low + high ) / 2;
      if( a[ mid ].compareTo( x ) < 0 )
        low = mid + 1;
      else if( a[ mid ].compareTo( x ) > 0 )
        high = mid - 1;
      else
        return mid;
    }
    return NOT_FOUND;    // NOT_FOUND = -1
  }

  // Test program
  public static void main( String [ ] args )
  {
    int SIZE = 8;
    Comparable [ ] a = new Integer [ SIZE ];
    for( int i = 0; i < SIZE; i++ )
      a[ i ] = new Integer( i * 2 );

    for( int i = 0; i < SIZE * 2; i++ )
      System.out.println( "Found " + i + " at " +
          binarySearch( a, new Integer( i ) ) );
  }
}
```