import java.util.Arrays;

/\*\*

 \* 顺序线性表的实现

 \*/

public class LineList<E>{

 private int size;   //长度

 private Object[] array;  //底层数组

 private final int default\_length=16; //默认长度

 /\*\*

  \* 无参构造方法

  \*/

 public LineList(){

  size = 0;

  //使用默认长度构造数组

  array = new Object[default\_length];

 }

 /\*\*

  \* 指定长度进行构造

  \* @param length 指定初始长度

  \*/

 public LineList(int length){

  if(length<0){

   throw new IllegalArgumentException("初始长度不合法:"+length);

  }

  //使用指定长度构造数组

  array = new Object[length];

 }

 /\*\*

  \* 指定初始化元素和长度进行构造

  \* @param element 初始化元素

  \* @param length 初始化长度

  \*/

 public LineList(E element,int length){

  if(length<1){

   throw new IllegalArgumentException("初始长度不合法:"+length);

  }

  //使用指定长度构造数组

  array = new Object[length];

  //初始化第一个元素

  array[0] = element;

  size++;

 }

 /\*\*

  \* 指定初始化元素进行构造

  \* @param element 初始化元素

  \*/

 public LineList(E element){

  //使用默认长度初始化数组

  array = new Object[default\_length];

  //初始化第一个元素

  array[0] = element;

 }

 /\*\*

  \* 获取元素个数

  \*/

 public int size() {

  return size;

 }

 /\*\*

  \* 判断是否为空

  \*/

 public boolean isEmpty() {

  return size==0;

 }

 /\*\*

  \* 判断是否包含此元素

  \*/

 public boolean contains(E e) {

  if(indexOf(e) == -1){

   return false;

  }

  return true;

 }

 /\*\*

  \* 格式化为数组

  \*/

 public Object[] toArray() {

  return Arrays.copyOf(array, size);

 }

 /\*\*

  \* 向线性表尾部添加一个元素

  \* @param e

  \* @return

  \*/

 public void add(E e) {

  extendCapacity(size+1);

  array[size]=e;

  size++;

 }

 /\*\*

  \* 扩容

  \* @param length 需要的长度

  \*/

 private void extendCapacity(int length){

  //当前数组长度和需要的长度取最大

  int minCapacity = Math.max(array.length, length);

  //判断是否需要扩容

  if(minCapacity - array.length>0){

   //数组长度增加一半

   int newLength = array.length + array.length/2;

   //如果新的长度还比需求要小,将需求的长度作为数组长度

   if(newLength < minCapacity){

    newLength=minCapacity;

   }

   //数组长度不能超过Integer.Max\_Value

   if(newLength > Integer.MAX\_VALUE - 8){

    newLength = Integer.MAX\_VALUE;

   }

   //数组扩容

   array = Arrays.copyOf(array, newLength);

  }

 }

 /\*\*

  \* 从线性表中移除所有此元素

  \* @param e 需要移除的元素

  \* @return

  \*/

 public void removeAll(E e) {

  if(e==null){

   for(int i=0;i<size;i++){

    if(array[i]==null){

     fastRemove(i);

    }

   }

  }else{

   for(int i=0;i<size;i++){

    if(e.equals(array[i])){

     fastRemove(i);

    }

   }

  }

 }

 /\*\*

  \* 删除索引处元素,后面的元素依次前移

  \* @param index 需要删除的索引

  \*/

 private void fastRemove(int index){

  if(size-index-1>0){

   //数组从index+1开始全部前移

   System.arraycopy(array, index+1, array, index, size-1);

  }

  //最后一个元素请空

  array[--size]=null;

 }

 /\*\*

  \* 清空线性表

  \*/

 public void clear() {

  //将数组全部填充为null

  Arrays.fill(array, null);

  //将元素个数改为0

  size=0;

 }

 /\*\*

  \* 获得索引处的元素

  \* @param index

  \* @return 索引处的元素

  \*/

 @SuppressWarnings("unchecked")

 public E get(int index) {

  checkIndex(index);

  return (E)array[index];

 }

 /\*\*

  \* 验证是否为索引越界

  \* @param index

  \*/

 private void checkIndex(int index){

  if(index>=size || index<0){

   throw new IndexOutOfBoundsException("索引越界");

  }

 }

 /\*\*

  \* 将索引处的元素修改为新的元素

  \* @param index 索引位置

  \* @param element 元素

  \* @return 原索引处的元素

  \*/

 @SuppressWarnings("unchecked")

 public E set(int index, E element) {

  checkIndex(index);

  E e = (E)array[index];

  array[index]=element;

  return e;

 }

 /\*\*

  \* 在指定的索引处插入指定的元素

  \* @param index 索引位置

  \* @param element 元素

  \*/

 public void add(int index, E element) {

  //验证索引

  checkIndex(index);

  //是否需要扩容

  extendCapacity(size+1);

  //复制数组

  System.arraycopy(array, index, array, index+1, size-index);

  array[index]=element;

 }

 /\*\*

  \* 移除索引处的元素

  \* @param index 索引

  \* @return 删除了的元素

  \*/

 @SuppressWarnings("unchecked")

 public E remove(int index) {

  checkIndex(index);

  //取得索引位置的元素

  E e = (E)array[index];

  fastRemove(index);

  return e;

 }

 /\*\*

  \* 取得元素第一次出现的位置的索引

  \* @param e 要查找的元素

  \* @return 如果为-1说明线性表没有这个元素

  \*/

 public int indexOf(E e) {

  if(e==null){

   for(int i=0;i<size;i++){

    if(e==array[i]){

     return i;

    }

   }

  }

  for(int i=0;i<size;i++){

   if(e.equals(array[i])){

    return i;

   }

  }

  return -1;

 }

 /\*\*

  \* 取得元素最后一次出现的位置的索引

  \* @param e 要查找的元素

  \* @return 如果为-1说明线性表没有这个元素

  \*/

 public int lastIndexOf(E e) {

  //判断元素是否为null

  if(e==null){

   for(int i=size-1;i>=0;i--){

    if(e==array[i]){

     return i;

    }

   }

  }

  for(int i=size-1;i>=0;i--){

   //如果为null这里会跑出NullPoint异常

   //所以前面要加上验证是否为Null

   if(e.equals(array[i])){

    return i;

   }

  }

  return -1;

 }

 /\*\*

  \* 截取线性表

  \* @param fromIndex 开始索引

  \* @param toIndex 结束索引

  \* @return 截取的线性表

  \*/

 @SuppressWarnings("unchecked")

 public LineList<E> subList(int fromIndex, int toIndex) {

  //判断开始索引是否越界

  if(fromIndex<0 || fromIndex >=size){

   throw new IndexOutOfBoundsException("开始索引越界:"+fromIndex);

  }

  //判断结束索引是否越界

  if(toIndex >=size || fromIndex <0){

   throw new IndexOutOfBoundsException("结束索引越界:"+toIndex);

  }

  //判断开始索引和结束索引是否正确

  if(fromIndex > toIndex){

   throw new IllegalArgumentException("参数不正确,开始索引应大于等于结束索引");

  }

  LineList<E> list = new LineList<E>();

  for(int i=fromIndex,j=toIndex;i<=j;i++){

   list.add((E)array[i]);

  }

  return list;

 }

}