

Konzepte objektorientierter Programmierung

Prof. Dr. Peter Müller

Werner Dietl

Software Component Technology

Exercises 9: Ownership

Wintersemester 06/07

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exercise 2 – ArrayList

```
public class ArrayList {
    private rep int[] array;
    private int next;
    public ArrayList() {
        array = new rep int[ 20 ];
        next = 0;
    }
    public void add( int i ) {
        if( next==array.length ) { resize( ); }
        array[ next ] = i;
        next++;
    }
}
```

Exercise 2 – ArrayList

```
public void setElems( readonly int[] ia ) {
    if( array.length < ia.length ) {
        array = new rep int[ ia.length ];    }
    for(int i=0; i<ia.length; ++i ) {
        array[i] = ia[i];    }
    next = ia.length;
}

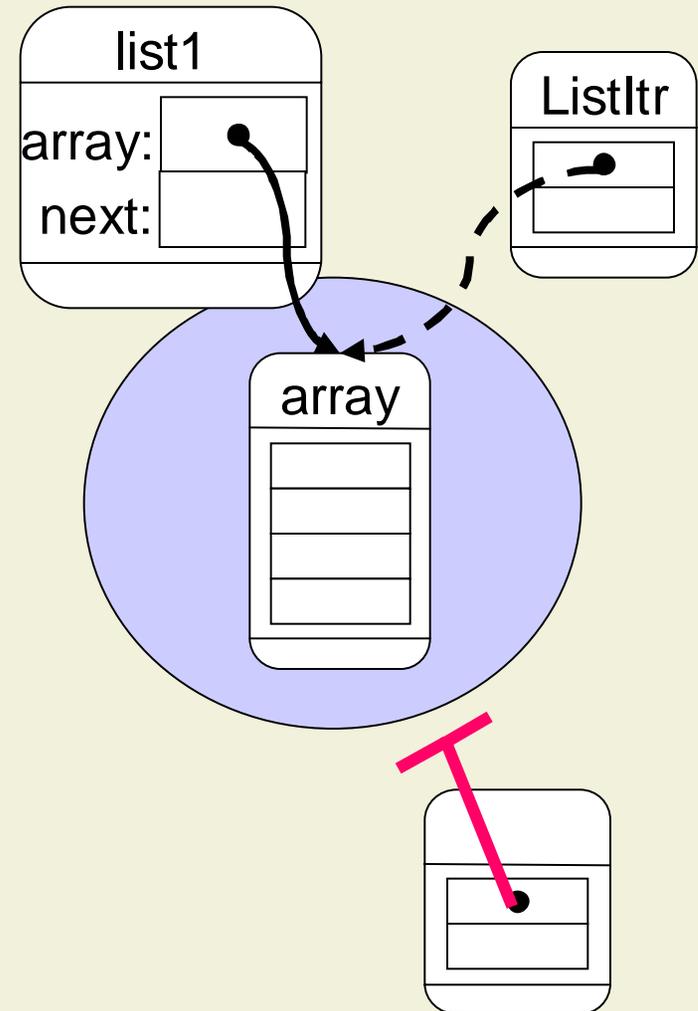
protected void resize() {
    if( next==array.length ) {
        readonly int[] oa = array;
        array = new rep int[2*oa.length];
        setElems( oa );    }
}
```

Exercise 2 – ArrayList

```
public readonly int[] getElems() {  
    return array; }  
  
public static void main( String[] args ) {  
    int[] myarr = new int[10];  
    for(int i=0; i<myarr.length; ++i)  
        myarr[i] = i;  
    ArrayList al = new ArrayList();  
    al.setElems( myarr );  
    System.out.println( "Before: " + al );  
    myarr[0] = 42;  
    System.out.println( "After:  " + al ); }  
}
```

Exercise 2 – ArrayList

- Array is encapsulated
- Readonly access possible
- No external modifications



Exercise 3 – Entries

```
class Entry {  
    readonly Object element;  
    peer Entry previous, next;  
  
    Entry( readonly Object o,  
          peer Entry p, peer Entry n ) {  
        element = o;  
        previous = p;  
        next = n; }  
}
```

Exercise 3 – LinkedList

```
public class LinkedList {  
    private rep Entry header;  
    private int size;  
  
    public LinkedList() {  
        header =  
            new rep Entry(null, null, null);  
        header.next = header;  
        header.previous = header;  
        size = 0;  
    }  
}
```

Exercise 3 – LinkedList

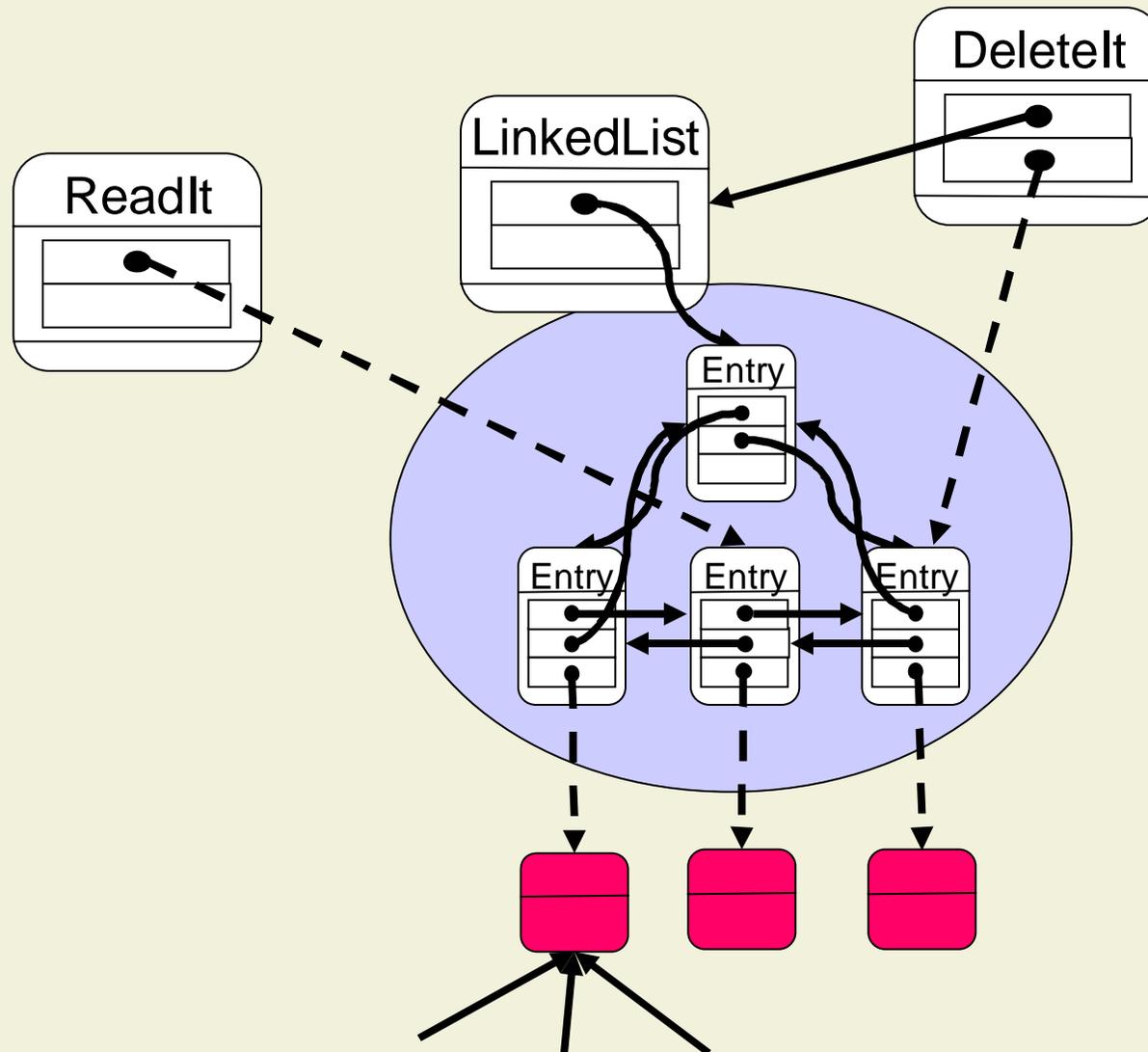
```
public void add( readonly Object o ) {  
    rep Entry newE =  
        new rep Entry(o, header, header.next);  
    header.next.previous = newE;  
    header.next = newE;  
    ++size;  
}
```

Exercise 3 – LinkedList

```
public pure readonly Object
    get( int idx ) {
    if( idx > size ) return null;

    readonly Entry e = header.next;
    for( int i=0; i<idx; ++i ) {
        e = e.next; }
    return e.element;
}
```

Exercise 3 – LinkedList



Exercise 3 – LinkedList

```
public pure peer ReadIterator
  getReadIterator() {
    return new peer ReadIterator(
      header );
  }

public pure peer DeleteIterator
  getDeleteIterator() {
    return new peer DeleteIterator(
      this, header );
  }
```

Exercise 3 – LinkedList

```
// precondition that the node belongs to us
/*@ requires e.owner == this; @*/
protected void delete( readonly Entry e ) {
    rep Entry re = (rep Entry) e;
    if( re.previous != null )
        re.previous.next = re.next;
    if( re.next != null )
        re.next.previous = re.previous;
    --size;
}
```

Exercise 3 – ReadIterator

```
public class ReadIterator {  
    public ReadIterator(readonly Entry h)  
    {  
        // the header is a dummy that is  
        // never null  
        current = h;  
        header = h;  
    }  
    public pure boolean hasNext() {  
        return current.next != header; }  
}
```

Exercise 3 – ReadIterator

```
public void moveNext() {  
    current = current.next;  
}
```

```
public pure readonly Object element() {  
    return current.element;  
}
```

```
protected readonly Entry current;  
protected readonly Entry header;  
}
```

Exercise 3 – DeleterIterator

```
public class DeleterIterator
    extends ReadIterator {
    public DeleterIterator( peer LinkedList l,
                          readonly Entry h ) {
        super( h ); list = l;
    }
    public void delete() {
        list.delete(current);
        current = current.next;
    }
    private peer LinkedList list;
}
```

Exercise 3 – LinkedList Main

```
public static void main( String[] args ) {  
    LinkedList ll = new LinkedList();  
    ll.add( new Integer(20) );  
    ll.add( "xyz" );  
    ll.add( new Float(2.2f) );  
    ll.add( "Hello World" );  
    ll.add( new Object() );  
    ReadIterator itr = ll.getReadIterator();  
    int i = 0;  
    while( itr.hasNext() ) { itr.moveToNext();  
        System.out.println("Element[" +i+ "]: " +  
            itr.element() );  
        ++i;  
    }  
}
```

Exercise 3 – LinkedList Main

```
DeleteIterator itd = ll.deleteIterator();
itd.moveNext();
itd.moveNext();
itd.delete();
System.out.println("Deleted second element.");
itr = ll.readIterator();
i = 0;
while( itr.hasNext() ) { itr.moveNext();
    System.out.println("Element[" + i + "]: " +
        itr.element() );
    ++i;
}
```

Exercise 3 – LinkedList

- Entries are encapsulated
- Readonly access possible
- No external modifications
- Iterator has to delegate the modification to the LinkedList
- LinkedList is in control

