03 A: Lists, Stacks, and Queues II

CS1102S: Data Structures and Algorithms

Martin Henz

January 29, 2010

Generated on Friday 29th January, 2010, 09:51

The Top-level Collection Interface

```
public interface Collection<Any>
        extends Iterable<Any>
{
    int size();
    boolean isEmpty();
    void clear();
    boolean contains(Any x);
    boolean add(Any x);          // sic
    boolean remove(Any x);       // sic
    java.util.Iterator<Any> iterator();
}
```

Iterable Objects Provide Iteration Pattern

```
public interface Iterator<Any> {
  boolean hasNext( );
  Any next( );
  void remove( );
}

while( itr.hasNext()) {
  Any item = itr.next( );
  System.out.println( item );
  }
}
```

Java Compiler Support for Iterators

```
for( Any item : coll )
    System.out.println( item );
```

becomes

```
while( itr.hasNext()) {
  Any item = itr.next( );
  System.out.println( item );
} }
```

The List Interface in Collection API

```
public interface List<Any>
        extends Collection<Any>
{
  Any get(int idx);
  Any set(int idx, Any newVal);
  void add(int idx, Any x);
  void remove(int idx);

  ListIterator<Any> listIterator(int pos);
}
```

ArrayList and LinkedList

```
public class ArrayList<Any>
       implements List<Any> {...}
public class LinkedList<Any>
       implements List<Any> {...}
```

Example: Remove Even Elements

Task

In a given list of Integer, remove all even integers, without copying the list (*in-place* operation)

```
ArrayList<Integer> myArrayList = ...;
LinkedList<Integer> myLinkedList = ...;
removeEvens(myArrayList);
removeEvens(myLinkedList);
```

ADT in Action

```
ArrayList<Integer> myArrayList = ...;
LinkedList<Integer> myLinkedList = ...;
removeEvens(myArrayList);
removeEvens(myLinkedList);
```

Observation

Both ArrayList and LinkedList implement the interface List. We can define removeEvens(...) in terms of List operations!

Inside and Outside

The same function removeEvens behaves differently for myLinkedList than for myArrayList!

In Detail: First Version

```
public static void removeEvensVer1(
                          List<Integer> lst) {
  int i = 0;
  while( i < lst.size( ) )
    if( lst.get( i ) % 2 == 0 )
      lst.remove( i );
    else
      i++;
}
```

In Detail: First Version

```
public static void removeEvensVer1(
                          List<Integer> lst) {
  int i = 0;
  while( i < lst.size( ) )
    if( lst.get( i ) % 2 == 0 )
      lst.remove( i );
    else
      i++;
}
```

Runtime for removeEvensVer1(myArrayList):

Runtime for removeEvensVer1(myLinkedList):

In Detail: First Version

```
public static void removeEvensVer1(
                          List<Integer> lst) {
  int i = 0;
  while( i < lst.size( ) )
    if( lst.get( i ) % 2 == 0 )
      lst.remove( i );
    else
      i++;
}
```

Runtime for removeEvensVer1(myArrayList): $O(N^2)$
Runtime for removeEvensVer1(myLinkedList):

In Detail: First Version

```
public static void removeEvensVer1(
                          List<Integer> lst) {
   int i = 0;
   while( i < lst.size( ) )
      if( lst.get( i ) % 2 == 0 )
         lst.remove( i );
      else
         i++;
}
```

Runtime for removeEvensVer1(myArrayList): $O(N^2)$
Runtime for removeEvensVer1(myLinkedList): $O(N^2)$

In Detail: Second Version

Idea

Use an iterator to go through the list, and remove element when found to be even

```
public static void removeEvensVer2(
                        List<Integer> lst) {
  for( Integer x : lst )
    if( x % 2 == 0 )
      lst.remove( x );
}
```

In Detail: Second Version

Idea

Use an iterator to go through the list, and remove element when found to be even

```
public static void removeEvensVer2(
                        List<Integer> lst) {
  for( Integer x : lst )
    if( x % 2 == 0 )
      lst.remove( x );
}
```

Runtime for removeEvensVer2(myArrayList): runtime error!
Runtime for removeEvensVer2(myLinkedList): runtime error!

In Detail: Third Version

Idea

Use the iterator's remove operation!

```
public static void removeEvensVer3(
                          List<Integer> lst) {
  Iterator<Integer> itr = lst.iterator();
  while(itr.hasNext())
    if(itr.next() % 2 == 0)
      itr.remove();
}
```

In Detail: Third Version

Idea

Use the iterator's remove operation!

```
public static void removeEvensVer3(
                          List<Integer> lst) {
  Iterator<Integer> itr = lst.iterator();
  while(itr.hasNext())
    if(itr.next() % 2 == 0)
      itr.remove();
}
```

Runtime for removeEvensVer3(myArrayList):

In Detail: Third Version

Idea

Use the iterator's remove operation!

```
public static void removeEvensVer3(
                          List<Integer> lst) {
  Iterator<Integer> itr = lst.iterator();
  while(itr.hasNext())
    if(itr.next() % 2 == 0)
      itr.remove();
}
```

Runtime for removeEvensVer3(myArrayList): $O(N^2)$

In Detail: Third Version

Idea

Use the iterator's remove operation!

```
public static void removeEvensVer3(
                          List<Integer> lst) {
  Iterator<Integer> itr = lst.iterator();
  while(itr.hasNext())
    if(itr.next() % 2 == 0)
      itr.remove();
}
```

Runtime for removeEvensVer3(myArrayList): $O(N^2)$
Runtime for removeEvensVer3(myLinkedList):

In Detail: Third Version

Idea

Use the iterator's remove operation!

```
public static void removeEvensVer3(
                        List<Integer> lst) {
  Iterator<Integer> itr = lst.iterator();
  while(itr.hasNext())
    if(itr.next() % 2 == 0)
      itr.remove();
}
```

Runtime for removeEvensVer3(myArrayList): $O(N^2)$
Runtime for removeEvensVer3(myLinkedList): $O(N)$

Our Implementation of ArrayList: MyArrayList

Highlights

- Maintain private array, capacity and current number of elements
- List operation `add` resizes automatically
- `get` and `set` use array directly
- `Iterator` provided

The Code

```
public class MyArrayList<Any>
       implements Iterable<Any>
{
  private static final int DEFAULT_CAPACITY = 10;
  private int theSize;
  private Any [ ] theItems;
  public MyArrayList( ) {
    clear( ); }
  public void clear( ) {
    theSize = 0;
    ensureCapacity( DEFAULT_CAPACITY );
  }
  ...
```

The Code

```
public class MyArrayList<Any> ...
{
   ...
   public int size() {
     return theSize; }
   public boolean isEmpty() {
     return size( ) == 0; }
   ...
}
```

The Code

```
public class MyArrayList<Any> ...
{
  ...
  public Any get( int idx ) {
    if( idx < 0 || idx >= size())
      throw new ArrayIndexOutOfBoundsException( );
    return theItems[ idx ];
  }
```

The Code

```
public class MyArrayList<Any> ...
{
  ...
  public Any set(int idx, Any newVal) {
    if( idx < 0 || idx >= size())
      throw new ArrayIndexOutOfBoundsException();
    Any old = theItems[idx];
    theItems[idx] = newVal;
    return old;
  }
  ...
```

The Code

```
public class MyArrayList<Any> ...
{
   ...
   public void ensureCapacity(int newCapacity) {
      if( newCapacity < theSize )
         return;
      Any [ ] old = theItems;
      theItems = (Any []) new Object[ newCapacity ];
      for( int i = 0; i < size( ); i++ )
         theItems[ i ] = old[ i ];
   }
   ...
}
```

The Code

```
public class MyArrayList<Any> ... {
  ...
  public boolean add(Any x) {
    add( size(), x);
    return true; }
  public void add(int idx, Any x) {
    if( theItems.length == size( ) )
      ensureCapacity( size( ) * 2 + 1 );
    for( int i = theSize; i > idx; i-- )
      theItems[ i ] = theItems[ i - 1 ];
    theItems[ idx ] = x;
    theSize++; }
  ...
}
```

The Code

```
public class MyArrayList<Any> ...
{
   ...
   public Any remove(int idx)
   {
      Any removedItem = theItems[ idx ];
      for( int i = idx; i < size( ) - 1; i++ )
         theItems[ i ] = theItems[ i + 1 ];

      theSize--;
      return removedItem;
   }
   ...
}
```

Excursion: Nested and Inner Classes

- Nested classes are classes that appear within other classes
- Visibility:
  - Nested class can use private fields and methods of surrounding class
  - Nested private class cannot be used within but not outside of surrounding class

Example of Nested Class

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
}
```

Excursion: Nested and Inner Classes

- Inner classes are non-static nested classes
- Instances of inner classes are created inside of instances of the surrounding class
- Methods of inner classes have access to non-static fields of the surrounding class

Example of Inner Class

```
class OuterClass {
  private int privateOuter;
  private class InnerClass {
    public int f() {
      return 2 * privateOuter;
  } }
  public int g() {
    InnerClass i = this.new InnerClass();
    return i.f();
} }
```

Inner Class in Action

```
public class MyArrayList<Any> ...
{
   ...
   public java.util.Iterator<Any> iterator( )
     { return new ArrayListIterator( ); }
   private class ArrayListIterator
           implements java.util.Iterator<Any>
   {
      ...
   }
}
```

Inner Class in Action

```
private class ArrayListIterator
        implements java.util.Iterator<Any>
  private int current = 0;
  public boolean hasNext() {
    return current < size( ); }
  public Any next() {
    if( !hasNext( ) )
      throw new java.util.NoSuchElementException();
    return theItems[ current++ ]; }
  public void remove() {
    MyArrayList.this.remove( --current ); }
} }
```

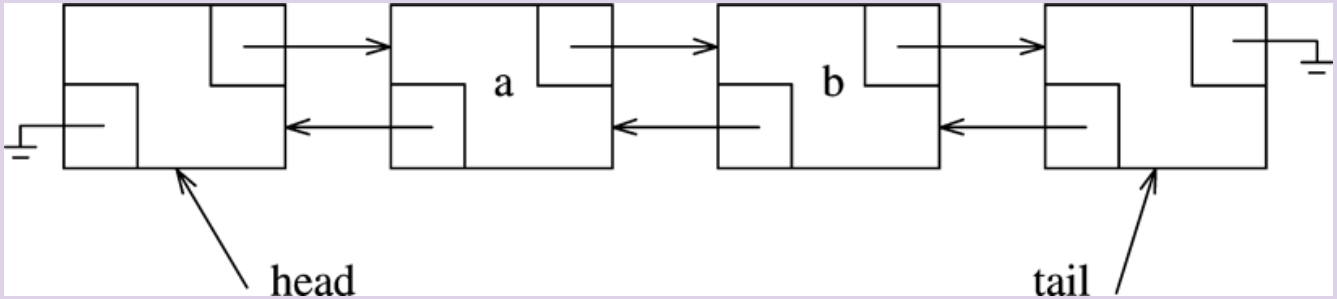
Our Implementation of LinkedList: MyLinkedList

Highlights

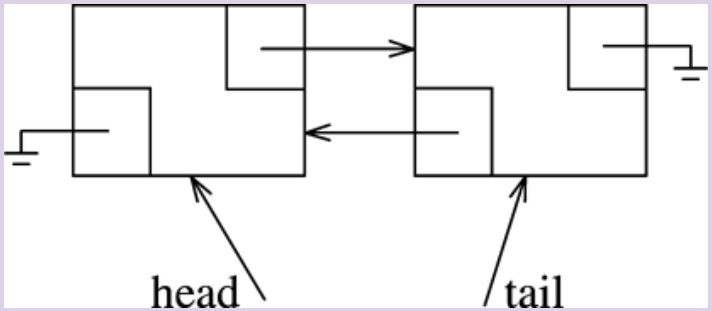
- Doubly-linked list, using `beginMarker` and `endMarker`
- Nodes are described by nested class
- Iterator is inner class like for `MyArrayList`

Some LinkedLists in Graphic Notation

A typical scenario



An empty list



The Code

```
public class MyLinkedList<Any>
        implements Iterable<Any> {
    private static class Node<Any>
      { ... }
    public MyLinkedList( )
      { clear( ); }
    public void clear( )
      { ... }
    public int size( )
      { return theSize; }
    public boolean isEmpty( )
      { return size( ) == 0; }
...}
```

The Code

```
private static class Node<Any> {
  public Node(Any d,Node<Any> p,Node<Any> n){
    data = d; prev = p; next = n; }
    public Any data;
    public Node<Any>   prev;
    public Node<Any>   next;
} }
```

The Code

```
public class MyLinkedList<Any> ...
{ ...
  public boolean add( Any x ) {
    add( size( ), x );  return true; }
  public void add( int idx, Any x ) {
    addBefore( getNode( idx ), x ); }
  public Any get( int idx ) {
    return getNode( idx ).data; }
  public Any set( int idx, Any newVal ) {
    Node<Any> p = getNode( idx );
    Any oldVal = p.data;
    p.data = newVal;
    return oldVal;
} }
```

The Code

```
public class MyLinkedList<Any> ...
{   ...
    public Any remove( int idx )
      { return remove( getNode( idx ) ); }
    private void addBefore( Node<Any> p, Any x )
      { ... }
    private Any remove( Node<Any> p )
      { ... }
    private Node<Any> getNode( int idx )
      { ... }
```

The Code

```
public class MyLinkedList<Any> ...
{    ...
     public java.util.Iterator<Any> iterator( )
        { return new LinkedListIterator( ); }
     private class LinkedListIterator
                   implements java.util.Iterator<Any>
       { ... }
```

The Code

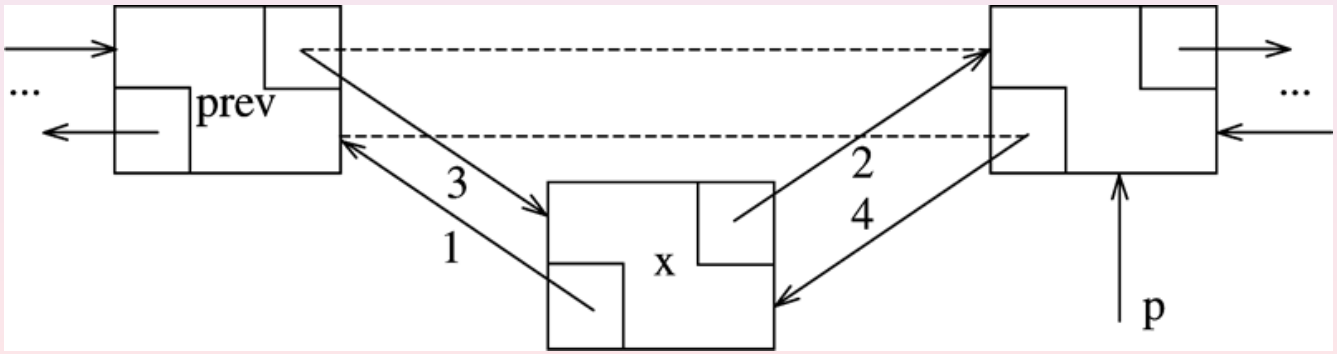
```
public class MyLinkedList<Any> ...
{    ...
     private int theSize;
     private int modCount = 0;
     private Node<Any> beginMarker;
     private Node<Any> endMarker;
}
```

The Code

```
public void clear( ) {
    beginMarker = new Node<Any>(null, null, null);
    endMarker = new Node<Any>(null, beginMarker, null);
    beginMarker.next = endMarker;
    theSize = 0;
    modCount++;
}
```

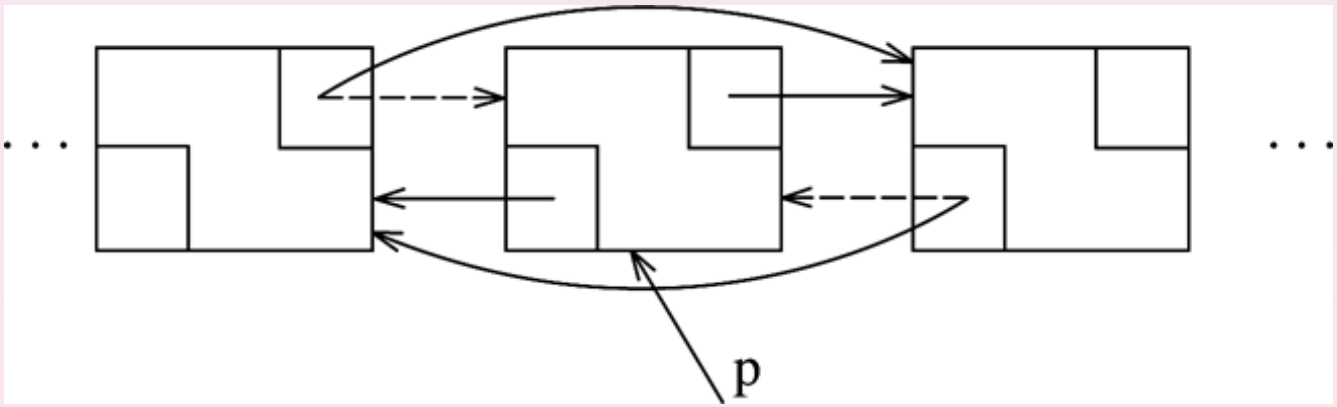
The Code

```
private void addBefore(Node<Any> p, Any x){
  Node<Any> newNode = new Node<Any>( x, p.prev, p );
  newNode.prev.next = newNode;
  p.prev = newNode;
  theSize++;
  modCount++; }
```



The Code

```
private Any remove( Node<Any> p ) {
  p.next.prev = p.prev;
  p.prev.next = p.next;
  theSize--;
  modCount++;
  return p.data; }
```



The Code

```
private Node<Any> getNode(int idx) {
  Node<Any> 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; }
```

The Code

```
private class LinkedListIterator
        implements java.util.Iterator<Any> {
  private Node<Any> current = beginMarker.next;
  private int expectedModCount = modCount;
  private boolean okToRemove = false;
  public boolean hasNext( ) {
    return current != endMarker;
  }
  ...
}
```

The Code

```
private class LinkedListIterator
        implements java.util.Iterator<Any> {
  ...
  public Any next( ) {
    if( modCount != expectedModCount )
      throw new java.util
        .ConcurrentModificationException();
    if( !hasNext( ) )
      throw new java.util.NoSuchElementException();
    Any nextItem = current.data;
    current = current.next;
    okToRemove = true;
    return nextItem; }
```

The Code

```
private class LinkedListIterator
        implements java.util.Iterator<Any> {
  ...
  public void remove( ) {
    if( modCount != expectedModCount )
      throw new java.util
        .ConcurrentModificationException( );
    if( !okToRemove )
      throw new IllegalStateException( );
    MyLinkedList.false.remove( current.prev );
    okToRemove = true;
} }
```

Puzzler: Animal Farm

```
public class AnimalFarm {
   public static void main(String[] args) {
      final String pig = "length: 10";
      final String dog = "length: " + pig.length();
      System.out.println("Animals are equal: "
                         + pig == dog);
   }
}
```

Next Week

- Queues
- Stacks

Next Week

- Queues
- Stacks
- ...and lots more Java!