CS2040 Lecture Note #5B:
List ADT & Linked Lists

Abstraction of a list (cont.)

Lecture Note #3B: List ADT & Linked Lists

- **Objectives:**
  - Able to implement various variants of linked list

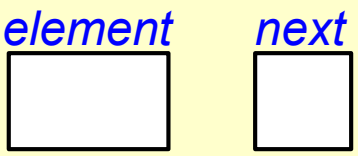
Outline

1. Use of a List (Motivation)
   - List ADT
2. List ADT Implementation via Array
   - Add and remove with an array
   - Time and space efficiency
3. List ADT Implementation via Linked Lists
   - Linked list approach
   - ListNode class: forming a linked list with ListNode
4. ADT of a Linked List
   - Various types of linked lists
   - BasicLinkedList
   - ExtendedLinkedList, TailedLinkedList, CircularLinkedList
   - DListNode class, DoublyLinkedList
5. Java class: LinkedList

Covered last week.

3.2 ListNode (using generic)

```
import java.util.*;

class ListNode <E> {
  protected E element;
  protected ListNode <E> next;

  /* constructors */
  public ListNode(E item) { element = item; next = null; }
  public ListNode(E item, ListNode <E> n) { element = item; next=n;}

  /* get the next ListNode */
  public ListNode <E> getNext() {
    return this.next;
  }

  /* get the element of the ListNode */
  public E getElement() {
    return this.element;
  }
}
```



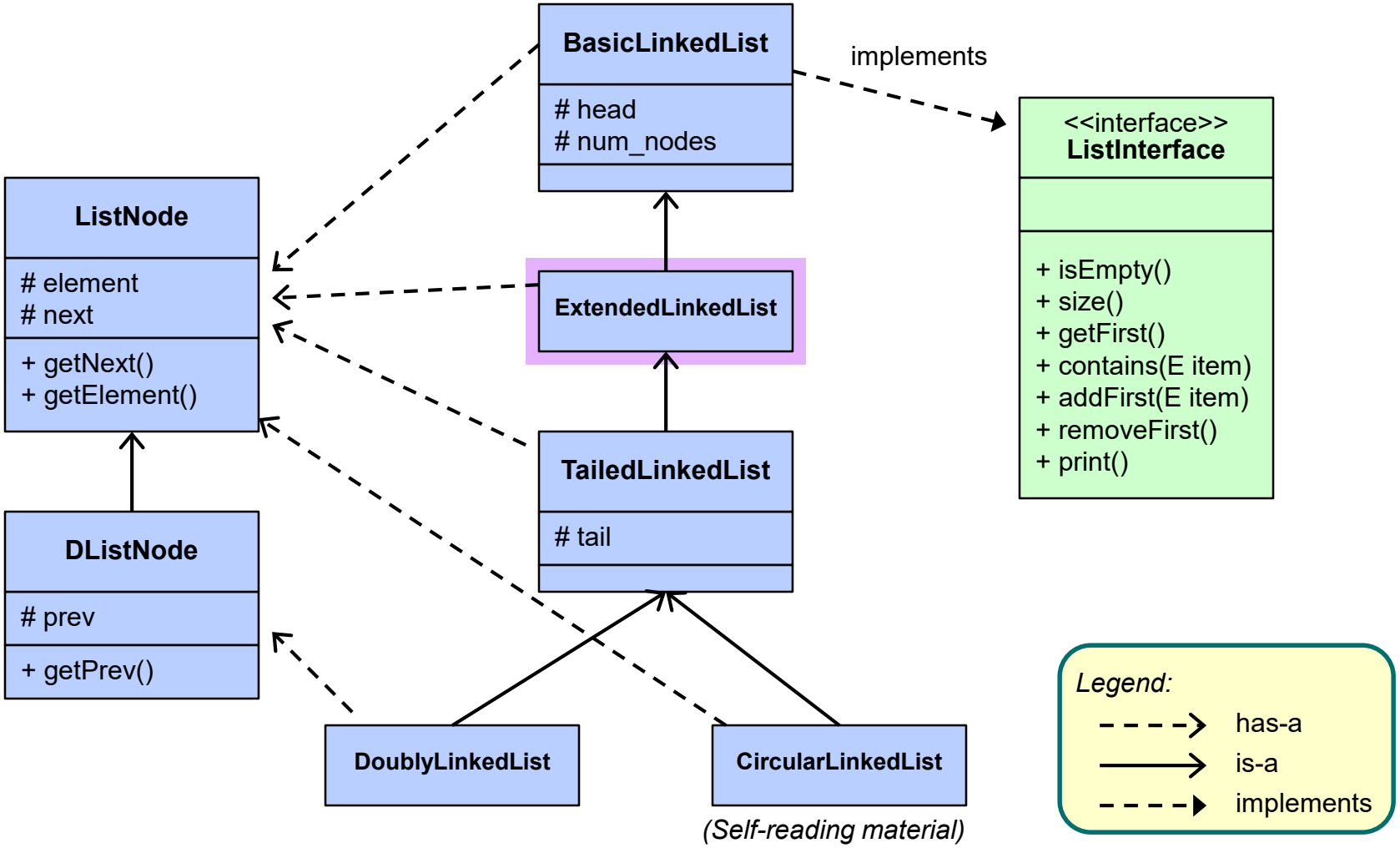
ListNode.java

4 Linked List ADT

Exploring linked list and its variants

4 Linked Lists: Variants

- We will continue with ExtendedLinkedList and the other variants now.



4.2 Extended Linked List (ELL) (1/11)

- When nodes are to be inserted to the middle of the linked list, BasicLinkedList (BLL) is not good enough.
- For example, if the items in the list must always be sorted according to some key values.
- In the discussion here, some of the methods from BLL are included to make the class complete.

4.2 Extended Linked List (2/11)

- An enhanced BasicLinkedList. Using the same ListNode class.

ExtendedLinkedList.java

```
import java.util.*;
class ExtendedLinkedList <E> implements ListInterface {
   private ListNode <E> head = null;
    private int num_nodes = 0;

    public boolean isEmpty() { return (num_nodes == 0); }
    public int size() { return num_nodes; }
    public E getFirst() throws NoSuchElementException {
        if (head == null)
            throw new NoSuchElementException("can't get from an empty list");
        else return head.getElement();
    }
    public boolean contains(E item) {
        for (ListNode <E> n = head; n!= null; n=n.next)
            if (n.getElement().equals(item)) return true;
        return false;
     }
```

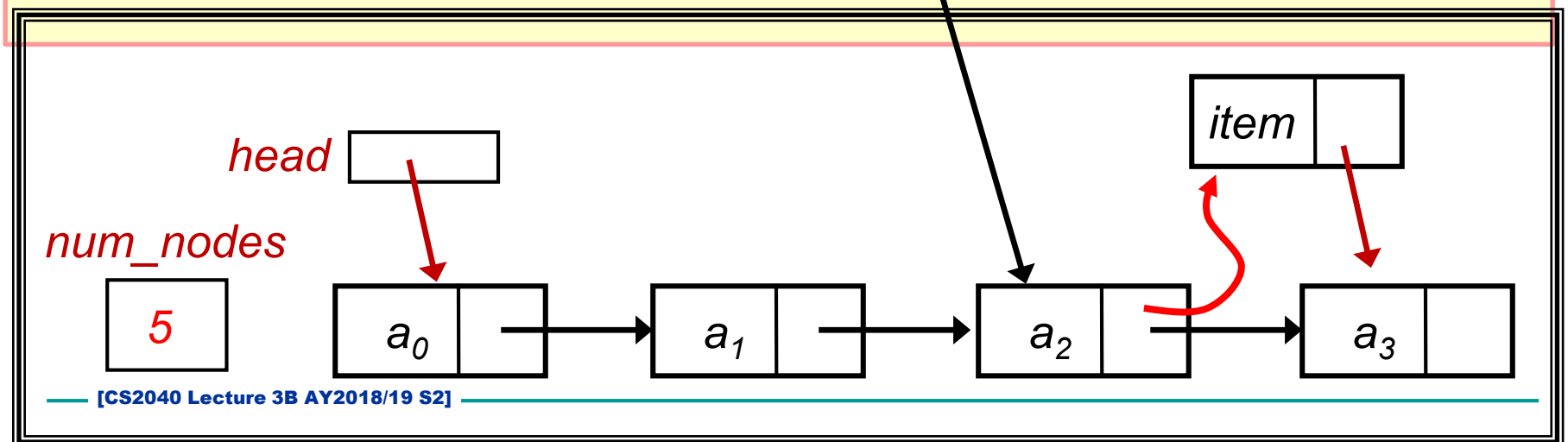
4.2 Extended Linked List (3/11)

```
public void addFirst(E item) {
    head = new ListNode <E> (item, head);
    num_nodes++;
}

public E removeFirst() throws NoSuchElementException {
     ListNode <E> ln;
     if (head == null)
        throw new NoSuchElementException("can't remove from empty list");
     else {
        ln = head;
        head = head.next;
        num_nodes--;
        return ln.element;
     }
 }
```

4.2 Extended Linked List (4/11)

ExtendedLinkedList.java

```
public ListNode <E> getFirstPtr() {
      return head;
}

public void addAfter(ListNode <E> current, E item) {
    if (current != null) {
        current.next = new ListNode <E> (item, current.next);
    } else {
        head = new ListNode <E> (item, head);
    }
     num_nodes++;
 }
```



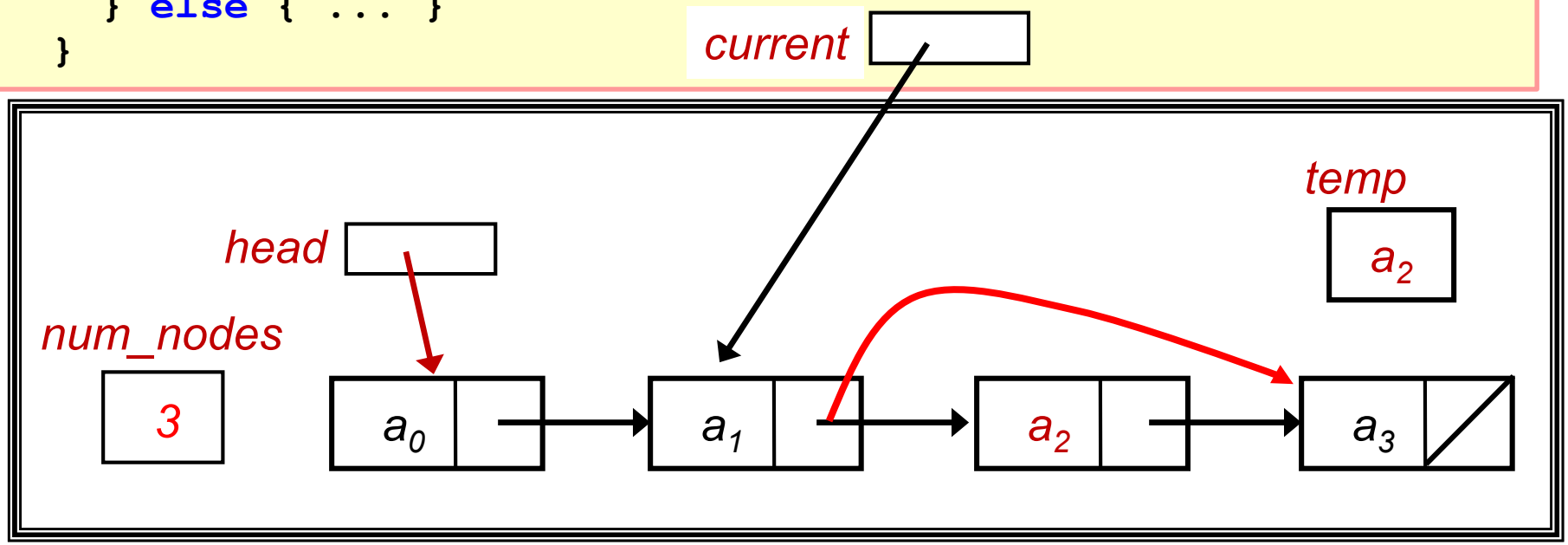
4.2 Extended Linked List (5/11)

ExtendedLinkedList.java

```
public E removeAfter(ListNode <E> current)
                    throws NoSuchElementException {
  E temp;
  if (current != null) {
    if (current.next != null) {// current is not pointing to last node
      temp = current.next.element;
      current.next = current.next.next;
      num_nodes--;  return temp;
    } else throw new NoSuchElementException("No next node to remove");
  } else { // if current is null, assume we want to remove head
    if (head != null) {
      temp = head.element;
      head = head.next;
      num_nodes--;  return temp;
    } else throw new NoSuchElementException("No next node to remove");
  }
}
 public void print() {
  ListNode <E> ln = head;
  System.out.print("List is: " + ln.element);
  for (int i=1; i < num_nodes; i++) {
    ln = ln.next;
    System.out.print(", " + ln.element);}
  System.out.println(".");
```

4.2 Extended Linked List (6/11)

```
public E removeAfter(ListNode <E> current) throws ... {
  E temp;
  if (current != null) {
    if (current.next != null) {
      temp = current.next.element;
      current.next = current.next.next;
      num_nodes--;
      return temp;
    } else throw new NoSuchElementException("...");
  } else { ... }
}
```



4.2 Extended Linked List (7/11)

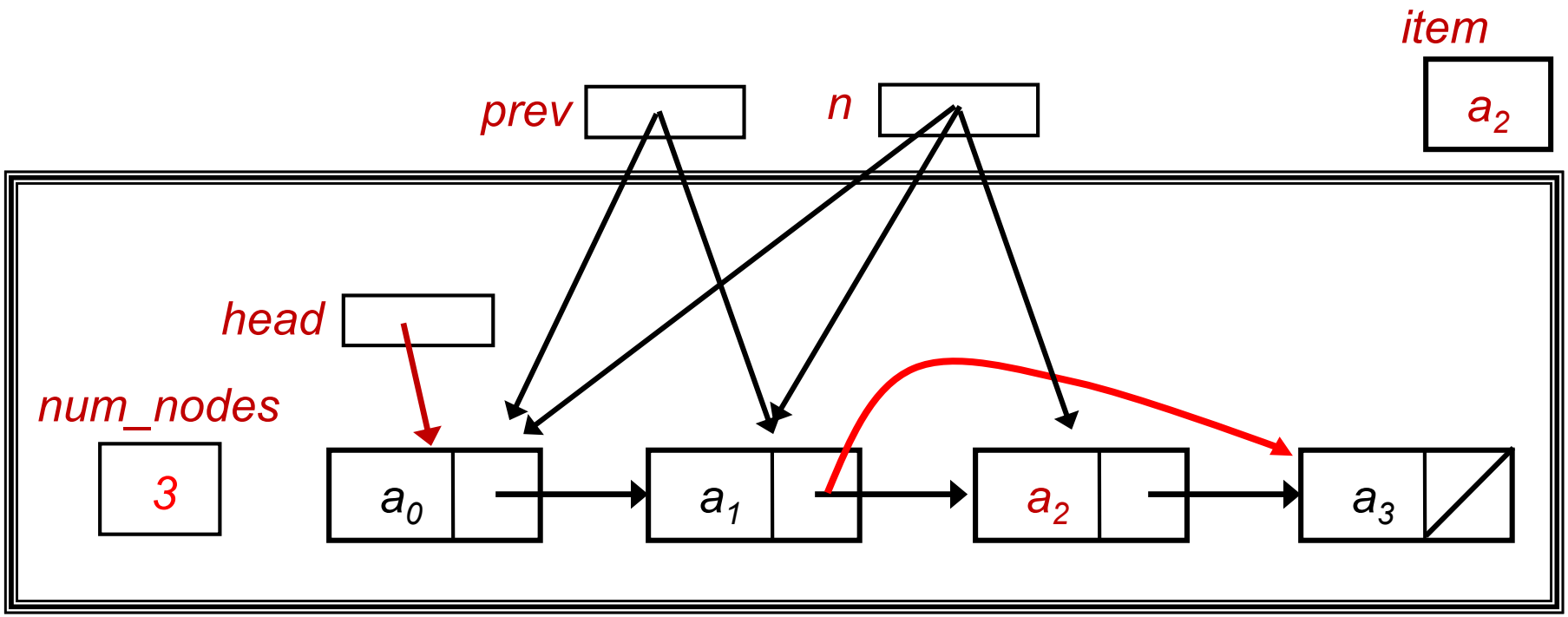
- To remove an item from a list, we first search for it and if it is found, use removeAfter() method to remove it.
- Consideration of cases are left to removeAfter()

ExtendedLinkedList.java

```
public E remove(E item)
                    throws NoSuchElementException {
  // Write your code below...









  }
}
```

4.2 Extended Linked List (8/11)

```
public E remove (E item) throws ... {



}
```



4.2 Extended Linked List (9/11)

- Example (Part 1 of 3)

ComplexCart.java

```
class ComplexCart implements Complex {
  private double real;
  private double imag;
  private final static double EPSILON = 0.00001;

  public ComplexCart(double r, double i) { real = r; imag = i; }

  public String toString() {
    if (imag == 0) return (realpart() + "");
    else if (imag < 0) return (real + "" + imag + "i");
    else return (real + "+" + imag + "i");
  }

  public boolean equals(Object cl) {
    if (cl instanceof Complex) {
      Complex temp = (Complex) cl;
      return (Math.abs(realpart() - temp.realpart()) < EPSILON &&
              Math.abs(imagpart() - temp.imagpart()) < EPSILON);
    }
    return false;
  }
  // Other code omitted
}
```

4.2 Extended Linked List (10/11)

- Example (Part 2 of 3)

TestExtended.java

```
import java.util.*;

class TestExtended {
  public static void main(String [] args) throws NoSuchElementException {

    ExtendedLinkedList <String> bl = new ExtendedLinkedList<String>();

    System.out.println("Part 1");
    bl.addFirst("aaa");
    bl.addFirst((new ComplexCart(2,3)).toString());
    bl.addFirst("ccc");
    bl.print();

    System.out.println();
    System.out.println("Part 2");
    ListNode <String> current = bl.getFirstPtr();
    bl.addAfter(current, "xxx");
    bl.addAfter(current, (new ComplexCart(6,6)).toString() );
    bl.print();
```

Answer?

4.2 Extended Linked List (11/11)

- Example (Part 3 of 3)

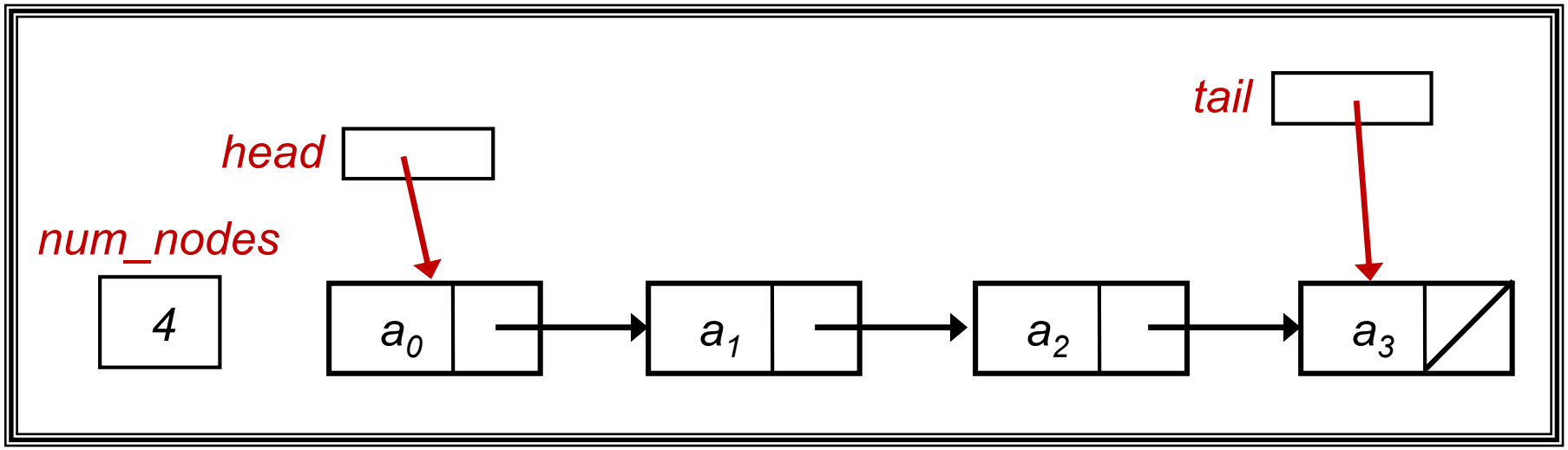
TestExtended.java

```
// (continue from slide 17)

    System.out.println();
    System.out.println("Part 3");
    if (bl.contains((new ComplexCart(3,3)).toString()))
      bl.remove((new ComplexCart(3,3)).toString());
    else
      System.out.println("complex no. does not exist");
    bl.print();

    System.out.println();
    System.out.println("Part 4");
    bl.removeAfter(current);
    bl.print();
  }
}
```

Answer?

4.3 Tailed Linked List (1/8)

- An enhanced Extended Linked List
  - To address the issue that adding to the end is slow
  - Add an extra data member called tail
  - Extra data member means extra maintenance too – no free lunch!
- Difficulty: Learn to take care of ALL cases of updating...



4.3 Tailed Linked List (2/8)

- An enhanced Extended Linked List (Part 1 of 5)
  - With a new data member: tail
  - Extra maintenance needed, see addFirst()

TailedLinkedList.java

```
import java.util.*;

class TailedLinkedList <E> {
  private ListNode <E> Head = null;
  private ListNode <E> tail = null;
  private int num_nodes = 0;
  public ListNode <E> getTail() {
     return tail;
  }
  public boolean isEmpty() { return (num_nodes == 0); }
  public int size() { return num_nodes; }
  public E getFirst() throws NoSuchElementException {
     if (head == null)
        throw new NoSuchElementException("can't get from an empty list");
     else return head.element;
  }
```

```
public void addFirst(E item) {
   head = new ListNode <E> (item, head);
   num_nodes++;
   if (num_nodes == 1)
      tail = head;
}
```

4.3 Tailed Linked List (3/8)

- An enhanced Extended Linked List (Part 2 of 5)
  - With the new member tail, can add to tail directly
  - Remember to update tail

```
public E getTail() throws NoSuchElementException {
    if (tail == null)
       throw new NoSuchElementException("empty list");
    else return tail.element;
}

public void addLast(E item) {
     if (head != null) {
        tail.next = new ListNode <E> (item);
        tail = tail.next;
        num_nodes++;
     } else {
        tail = new ListNode <E> (item);
        head = tail;
        num_nodes++;
     }
}
```

4.3 Tailed Linked List (4/8)

- An enhanced Extended Linked List (Part 3 of 5)
  - Make sure that the tail pointer is handled for all possible cases.

```
public void addAfter(ListNode <E> current, E item) {
if (current != null) {
        current.next = new ListNode <E> (item, current.next);
        if (current == tail)
            tail = current.next;
    } else {
            head = new ListNode <E> (item, head);
            if (tail == null) tail = head;
        }
    num_nodes++;
}
```

```
public void addFirst(E item) {
   // how to make use of addAfter here?
}
```

4.3 Tailed Linked List (5/8)

- An enhanced Extended Linked List (Part 4 of 5)
  - Can simply reuse parent's methods?

TailedLinkedList.java

```
public E removeAfter(ListNode <E> current)
                    throws NoSuchElementException {
    E temp;
    if (current != null) {
        if (current.next != null) {
            temp = current.next.element;
            current.next = current.next.next;
            num_nodes--;
            if (current.next == null) tail = current;
            return temp;
} else throw new NoSuchElementException("No next node to remove");
    } else { // if current is null, assume we want to remove head
      // continue next slide
```

4.3 Tailed Linked List (6/8)

- An enhanced Extended Linked List (Part 5 of 5)
  - Can simply reuse parent's methods?

TailedLinkedList.java

```
    // continue from previous slide
    if (head != null) {
            temp = head.element;
            head = head.next;
            num_nodes--;
            if (head == null) tail = null;
            return temp;
        } else throw new NoSuchElementException("No next node to remove");
}

public E removeFirst() throws NoSuchElementException {
  return removeAfter (null);
}
```

  - Other good methods to be in the class: removeLast()
  - Need to include other methods such as print() to make the class complete

4.3 Tailed Linked List (7/8)

- Example (Part 1 of 2)

TestTailed.java

```
import java.util.*;

class TestTailed {
  public static void main(String [] args) throws NoSuchElementException {

    TailedLinkedList <String> bl = new TailedLinkedList <String>();

    System.out.println("Part 1");
    bl.addFirst("aaa");
    bl.addFirst((new ComplexCart(2,3)).toString());
    bl.addFirst("ccc");
    bl.print();

    System.out.println();
    System.out.println("Part 2");
    ListNode <String> current = bl.getFirstPtr();
    bl.addAfter(current, "xxx");
    bl.addAfter(current, (new ComplexCart(6,6)).toString() );
    bl.print();
```

Answer?

4.3 Tailed Linked List (8/8)

- Example (Part 2 of 2)

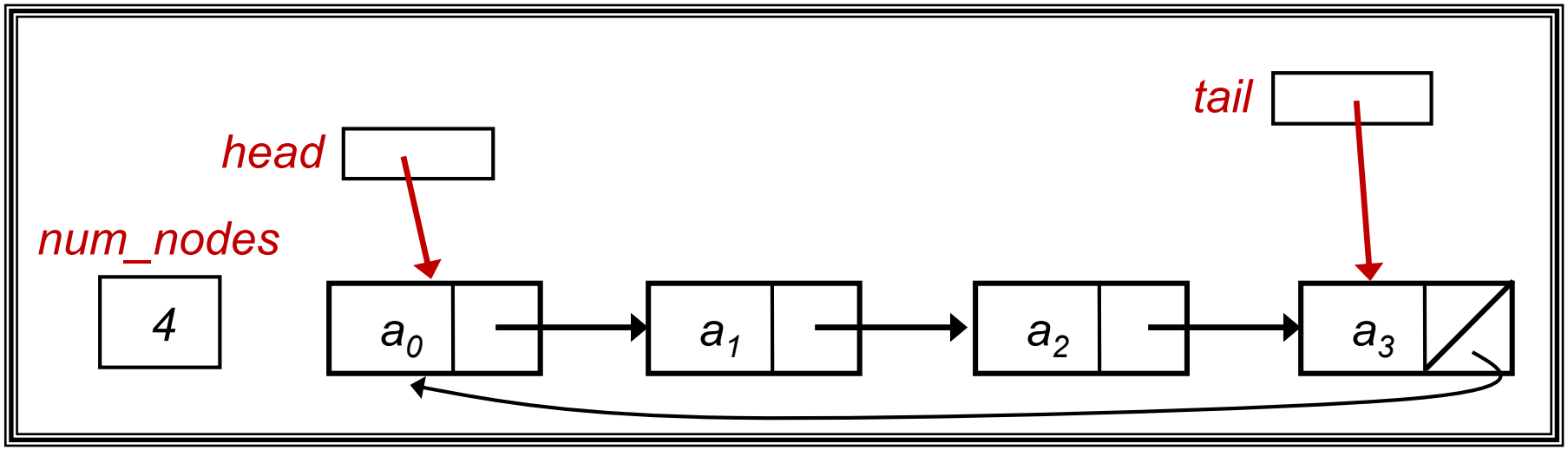
TestTailed.java

```
// (continue from slide 25)
    System.out.println();
    System.out.println("Part 3");
    if (bl.contains((new ComplexCart(2,3)).toString()))
      bl.remove((new ComplexCart(2,3)).toString());
    else
      System.out.println("complex no. does not exist");
    bl.print();

    System.out.println();
    System.out.println("Part 4");
    bl.removeAfter(current);
    bl.print();
  }
}
```

Answer?

4.4 Circular Linked List (1/9) [self-reading]

- An enhanced Tailed Linked List
  - To allow cycling through the list repeatedly, e.g. in a round robin system to assign shared resource
  - Add a link from tail node to point back to head node
  - Different in linking need different maintenance – no free lunch!
- Difficulty: Learn to take care of ALL cases of updating...



4.4 Circular Linked List (2/9) [self-reading]

- Yet another enhanced Extended Linked List

CircularLinkedList.java

```
import java.util.*;

class CircularLinkedList <E> {
  // codes of TailLinkedList here

  public void addFirst(E item) {
    // codes for addFirst() of TailLinkedList here
    tail.next = head;
  }

  public void addLast(E item) {
    // codes for addLast() of TailLinkedList here
    tail.next = head;
  }
```

4.4 Circular Linked List (3/9) [self-reading]

- Need to make sure it is circular

```
public void addAfter(ListNode <E> current, E item) {
    if (current != null) {
        current.next = new ListNode <E> (item, current.next);
        if (current == tail)
            tail = current.next;
    }else{
            head = new ListNode <E> (item, head);
            if (tail == null) // adding to empty list
               tail = head;
               tail.next = head;
          }
    num_nodes++;
}
```

4.4 Circular Linked List (4/9) [self-reading]

CircularLinkedList.java

```
public E removeAfter(ListNode <E> current)
                throws NoSuchElementException {
   E temp;
   if (current != null) {
      if (current.next == current) { // removing the only node
         temp = head.element;
         head = null;
         tail = null;
         num_nodes = 0;
         return temp;
      }
      else {
         temp = current.next.element;
         if (current.next == head)          // to remove head
             head = head.next;
         else if (current.next == tail)   // to remove tail
             tail = current;
         current.next = current.next.next;
         num_nodes--;
         return temp;
      }                          // continue on next slide
```

4.4 Circular Linked List (5/9) [self-reading]

- Yet another enhanced Extended Linked List
  - Can simply reuse parent's methods?

CircularLinkedList.java

```
      } else { // if current is null, assume we want to remove head
        if (head != null) {
            temp = head.element;
            head = head.next;
            tail.next = head;
            num_nodes--;
            if (num_nodes == 0) {
               head = null;
               tail = null;
            }
            return temp;
        } else throw new NoSuchElementException("No next node to
remove");
      }
  }

   public E removeFirst() throws NoSuchElementException {
      return removeAfter(null);
   }
```

4.4 Circular Linked List (6/9) [self-reading]

- Yet another enhanced Extended Linked List
  - Can simply reuse parent's methods?

CircularLinkedList.java

```
// the one in ExtendedLinkedList can't be used due to infinite loop
public E remove(E item) throws NoSuchElementException {
  if (head == null)
    throw new NoSuchElementException("can't find item");

  ListNode <E> n = head;
  ListNode <E> prev = null;
  for (int i=0; i < num_nodes; i++, prev=n, n=n.next)
    if (n.element.equals(item))
      return removeAfter(prev);

  throw new NoSuchElementException("can't find item to remove");
}
```

4.4 Circular Linked List (7/9) [self-reading]

- Yet another enhanced Extended Linked List
  - Can simply reuse parent's methods?

CircularLinkedList.java

```
  // the one in ExtendedLinkedList can't be used due to infinite loop
  public boolean contains(E item) throws NoSuchElementException {
    if (head == null) return false;
    ListNode <E> n = head;
    for (int i=0; i < num_nodes; i++) {
      if (n.element.equals(item)) { return true; }
      n = n.next;
    }
    return false;
  }
}
```

4.4 Circular Linked List (8/9) [self-reading]

- Example (Part 1 of 2)

TestCircular.java

```
import java.util.*;

class TestCircular {
  public static void main(String [] args) throws NoSuchElementException {

    CircularLinkedList <String> bl = new CircularLinkedList <String> ();

    System.out.println("Part 1");
    bl.addLast("aaa");
    bl.addLast((new ComplexCart(2,3)).toString());
    bl.addLast("ccc");
    bl.print();

    System.out.println();
    System.out.println("Part 2");
    ListNode <String> current = bl.getFirstPtr();
    bl.addAfter(current, "xxx");
    bl.addAfter(current, (new ComplexCart(6,6)).toString());
    bl.print();
```

Answer?

4.4 Circular Linked List (9/9) [self-reading]

- Example (Part 2 of 2)

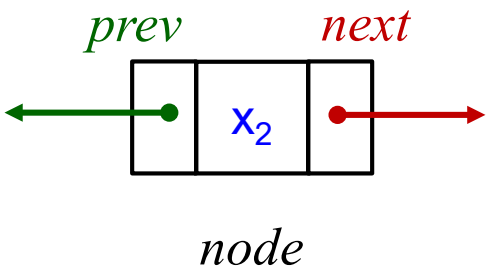
TestCircular.java

```
// (continue from slide 34)
    System.out.println();
    System.out.println("Part 3");
    if (bl.contains((new ComplexCart(3,3)).toString()))
      bl.remove((new ComplexCart(3,3)).toString());
    else
      System.out.println("complex no. does not exist");
    bl.print();

    System.out.println();
    System.out.println("Part 4");
    bl.removeAfter(null);
    bl.removeAfter(bl.getTail());
    bl.removeFirst();
    bl.print();
  }
}
```

Answer?

4.5 Doubly List Node (1/2)

- Motivation
  - In the preceding discussion, we have a “next” pointer to move forward
  - Often, we need to move backward as well
  - Use a “prev” pointer to allow backward traversal
  - Once again, no free lunch – need to maintain “prev” in all updating methods
  - We extend ListNode to get DListNode



4.5 Doubly List Node (2/2)

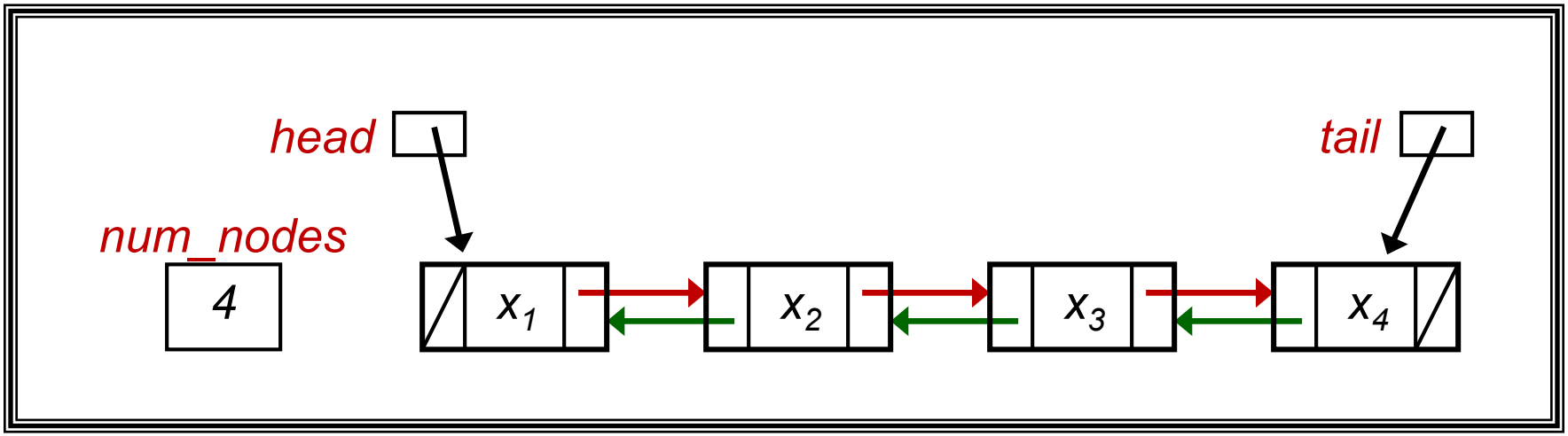
- DListNode to contain both “next” and “prev” pointers.

DListNode.java

```
class DListNode <E> {
  private E element;
  private DListNode <E> next;
  private DListNode <E> prev;

  public DListNode(E item, DListNode <E> n, DListNode <E>  p) {
     element = item; next = n; prev = p;
  }
  public DListNode(E item) {
      element = item; next = null; prev = null;
  }
  public E getElement(){return element;} // get the element
  public DListNode getNext(){return next;} // get the next DListNode
  public DListNode getPrev(){return prev;} // get the prev DListNode
}
```

4.6 Doubly Linked List (1/11)

- Use DListNode to have 2 pointers: “next” and “prev”
- Extend from TailedLinkedList class



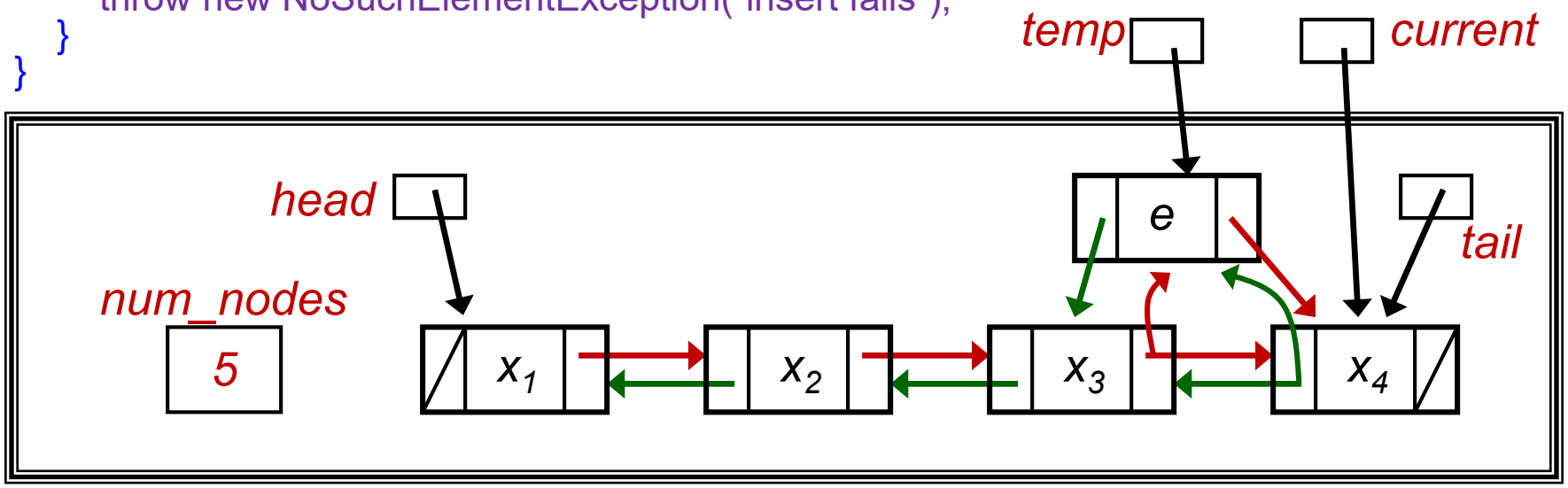
4.6 Doubly Linked List (2/11)

DoublyLinkedList.java

```
import java.util.*;

class DoublyLinkedList <E> {
  private DListNode head = null;

  public void addBefore(DListNode <E> current, E item)
                                        throws NoSuchElementException {
    if (current != null) {
      DListNode <E> temp =
                    new DListNode <E> (item, current, current.prev);
      num_nodes++;

      if (current != head) {
        current.prev.next = temp;
        current.prev = temp;
      } else {
        current.prev = temp;
        head = temp;
      }
    } else {
      throw new NoSuchElementException("insert fails");
    }
  }
```

4.6 Doubly Linked List (3/11)

```
public void addBefore (DListNode <E> current, E item) throws NoSuchElementException {

    if (current != null) {
        DListNode <E> temp = new DListNode <E> (item,current,current.prev);
        num_nodes++;

        if (current != head) {
            current.prev.next = temp;
            current.prev = temp;
        } else { // Insert node before head
            current.prev = temp;
            head = temp;
        }
    }  else { // If current is null, insertion fails.
        throw new NoSuchElementException("insert fails");
    }
}
```

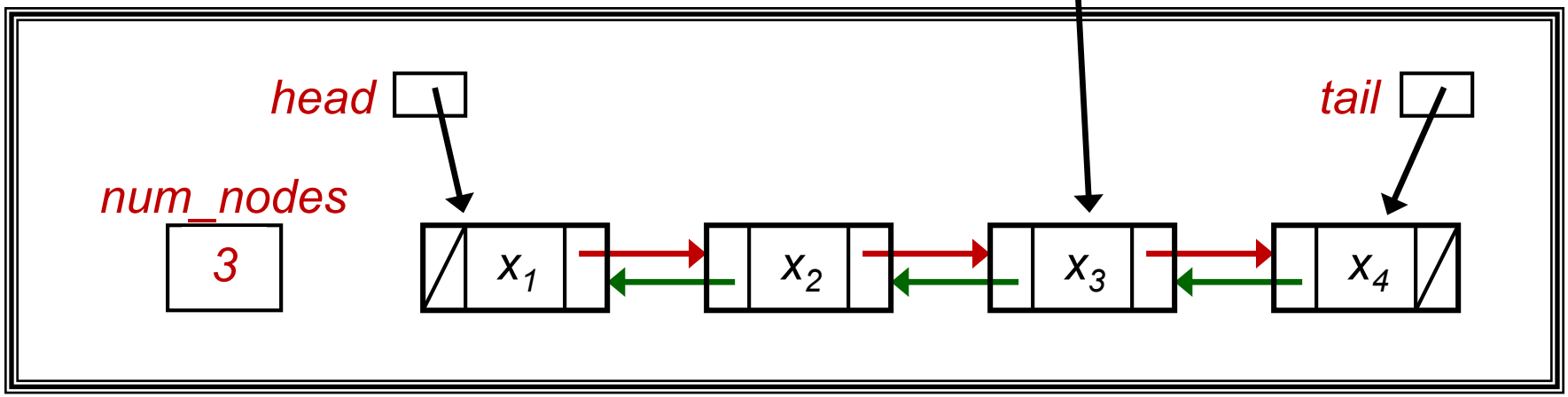


4.6 Doubly Linked List (5/11)

- To remove a node, fix both "next" and "prev" pointers (next 2 slides)

DoublyLinkedList.java

```
public E removeCurrent(DListNode <E> current)
                                    throws NoSuchElementException {
  if (current == null)
    throw new NoSuchElementException("remove fails");
  else
    num_nodes--;
  if (current == head && current == tail) {
    head = null;
    tail = null;
    return current.element;
  }
```

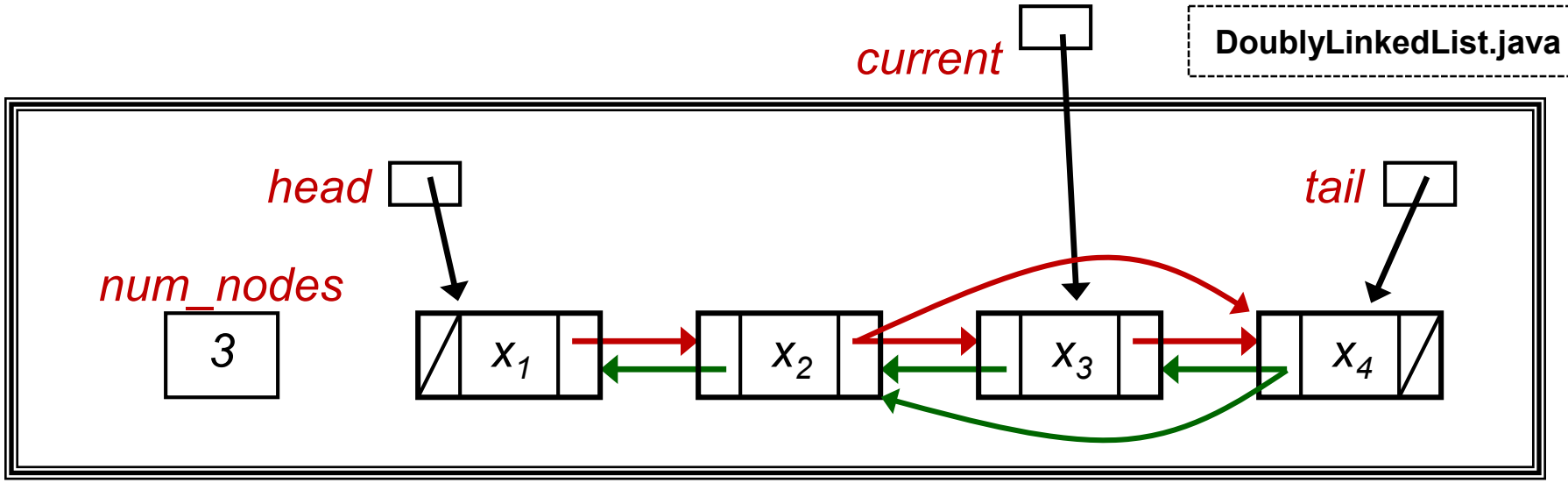


4.6 Doubly Linked List (6/11)

```
// fixing the prev pointer
if (current != tail) {
  current.next.prev=current.prev;
} else {
  // deleting tail
  tail = current.prev;
  tail.next = null;
  return current.element;
}
```

```
// fixing the next pointer
if (current != head)
  current.prev.next = current.next;
else {
  head = current.next;
  head.prev = null;
}
return current.element;
}
```

DoublyLinkedList.java



4.6 Doubly Linked List (8/11)

- Example (Part 1 of 2)

TestDoubly.java

```
import java.util.*;

class TestDoubly {
  public static void main(String [] args) throws NoSuchElementException {

    DoublyLinkedList <String> bl = new DoublyLinkedList <String> ();

    System.out.println("Part 1");
    bl.addFirst("aaa");
    bl.addFirst((new ComplexCart(2,3)).toString());
    bl.addFirst("ccc");
    bl.print();
    bl.printBackToFront();

    System.out.println();
    System.out.println("Part 2");
    DListNode <String> current = (DListNode <String>) bl.getFirstPtr();
    bl.addBefore(current, "xxx");
    bl.addBefore(current, (new ComplexCart(6,6)).toString());
    bl.print();
    bl.printBackToFront();
```

Answer?

4.6 Doubly Linked List (10/11)

- Example (Part 2 of 2)

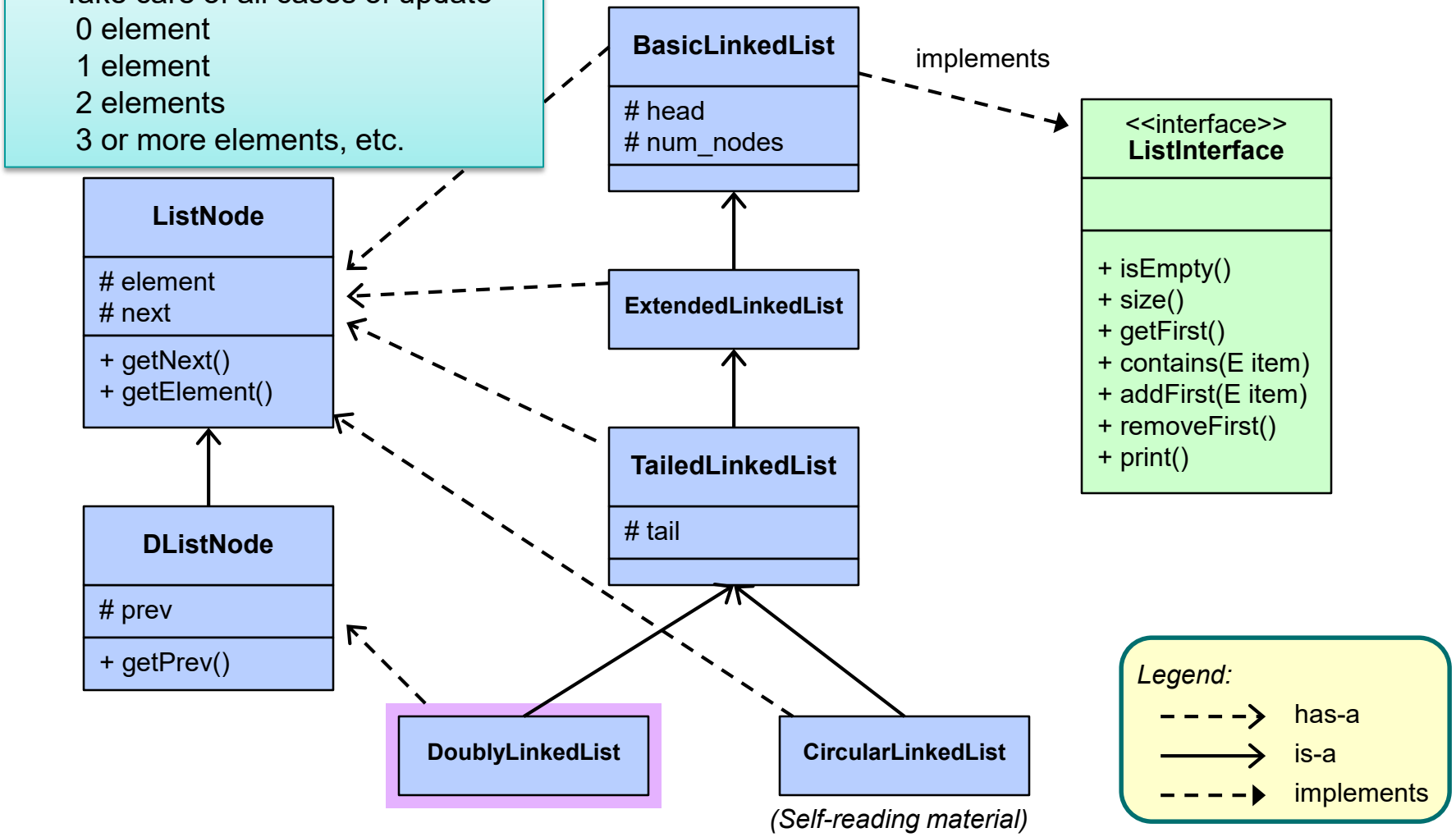
TestDoubly.java

```
    System.out.println();
    System.out.println("Part 3");
    bl.removeCurrent((DListNode <String>) current );
    if (bl.contains((new ComplexCart(2,3)).toString()))
      bl.remove((new ComplexCart(2,3)).toString());
    else
      System.out.println("complex no. does not exist");
    bl.remove("aaa");
    bl.removeFirst();
    bl.printBackToFront();

    System.out.println();
    System.out.println("Part 4");
    current = (DListNode <String>) bl.getFirstPtr();
    bl.removeAfter(current);
    bl.addAfter(current, (new ComplexCart(4,9)).toString());
    bl.printBackToFront();
  }
}
```

Answer?

4 Linked Lists: Variants

Difficulty: (Boundary cases)
Take care of all cases of update
0 element
1 element
2 elements
3 or more elements, etc.



5 Java Class: LinkedList

Using the LinkedList class

5 Java Class: LinkedList <E>

- This is the class provided by Java library
- This is the linked list implementation of the List interface
- It has many more methods than what we have discussed so far of our versions of linked lists. On the other hand, we created some methods not available in the Java library class too.
- Please do not confuse this library class from our versions in Section 4. In a way, we open up the Java library to show you the inside working.
- For purposes of sit-in labs or exam, please use whichever one as you are told if stated.

5.1 Class LinkedList: API

Constructor Summary

`LinkedList()`	Constructs an empty list.
`LinkedList(Collection c)`	Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Method Summary

void	`add(int index, Object element)` Inserts the specified element at the specified position in this list.
boolean	`add(Object o)` Appends the specified element to the end of this list.
boolean	`addAll(Collection c)` Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	`addAll(int index, Collection c)` Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	`addFirst(Object o)` Inserts the given element at the beginning of this list.
void	`addLast(Object o)` Appends the given element to the end of this list.
void	`clear()` Removes all of the elements from this list.

5.2 Class LinkedList (1/2)

- An example use (Page 1 of 2)

TestLinkedListClass.java

```
import java.util.*;

class TestLinkedListClass {
  static void printList(LinkedList <Integer> alist) {
    System.out.print("List is: ");
    for (int i = 0; i < alist.size(); i++)
      System.out.print(alist.get(i) + "\t");
    System.out.println();
  }

  static void printListv2(LinkedList <Integer> alist) {
    System.out.print("List is: ");
    while (alist.size() != 0) {
      System.out.print(alist.element() + "\t");
      alist.removeFirst();
    }
    System.out.println();
  }
```

5.2 Class LinkedList (2/2)

- An example use (Page 2 of 2)

TestLinkedListClass.java

```
 public static void main(String [] args) {
   LinkedList <Integer> alist = new LinkedList <Integer> ();
   for (int i = 1; i <= 5; i++)
     alist.add(new Integer(i));

   printList(alist);

   System.out.println("First element: " + alist.getFirst());
   System.out.println("Last element: "  + alist.getLast());
   alist.addFirst(888);
   alist.addLast(999);
   printListv2(alist);
   printList(alist);
 }
}
```

6 Summary (1/2)

- We learn to create our own data structure
  - In creating our own data structure, we face 2 difficulties:
    1. Manipulation of pointers/references (which one goes first)
    2. Careful with all the boundary cases
  - Drawings can often help in understanding the cases (point 2), which then can help in knowing what can be used/manipulated (points 1)
  - We may need to re-organize cases to combine into codes

6 Summary (2/2)

- Once we can get through this lecture, the rest should be smooth sailing as all the rest are similar in nature
  - You should try to add more methods to our versions of LinkedList, or to extend ListNode to other type of node
- Please do not forget that the Java Library class is much more comprehensive than our own – for sit-in labs and exam, please use whichever one as you are told if stated.