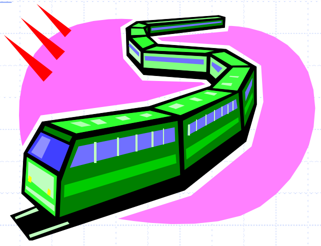
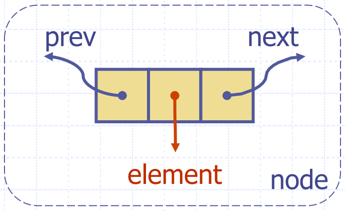
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

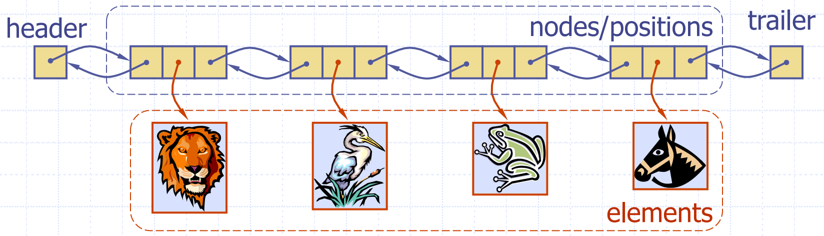
Doubly Linked Lists



Doubly Linked List

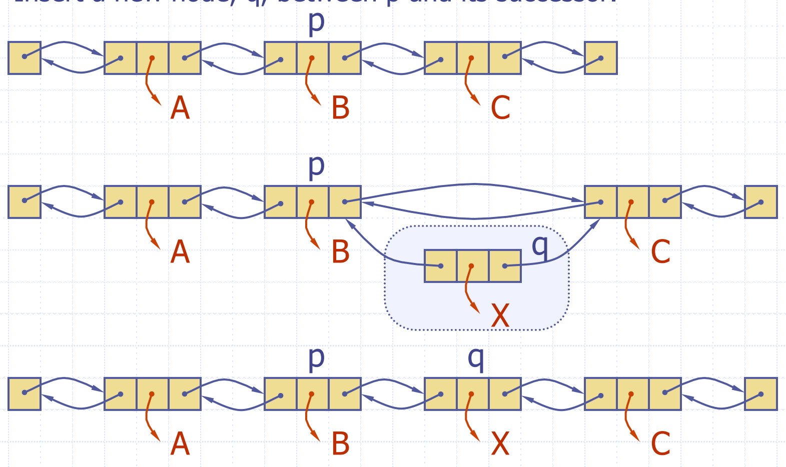
- A doubly linked list can be traversed forward and backward
- Nodes store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes





Insertion

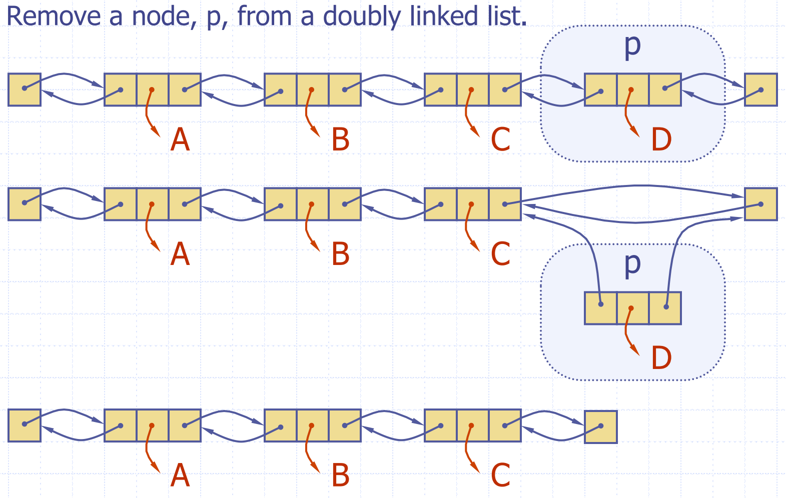
- Insert a new node, q, between p and its successor.



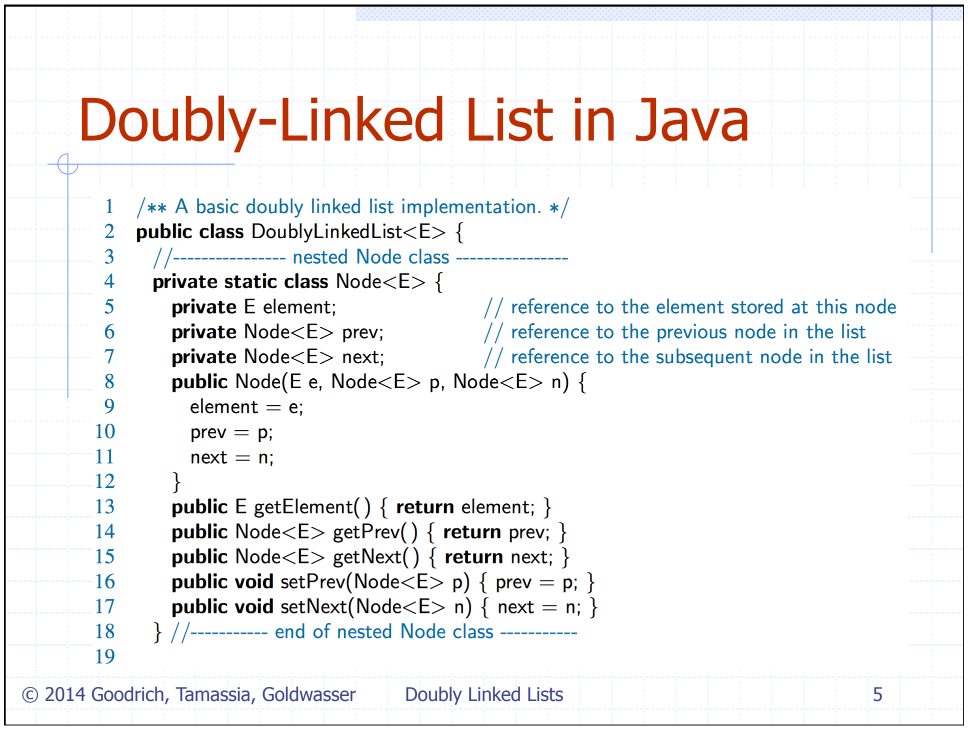
© 2014 Goodrich, Tamassia, Goldwasser Doubly Linked Lists 3

Deletion

- Remove a node, p, from a doubly linked list.



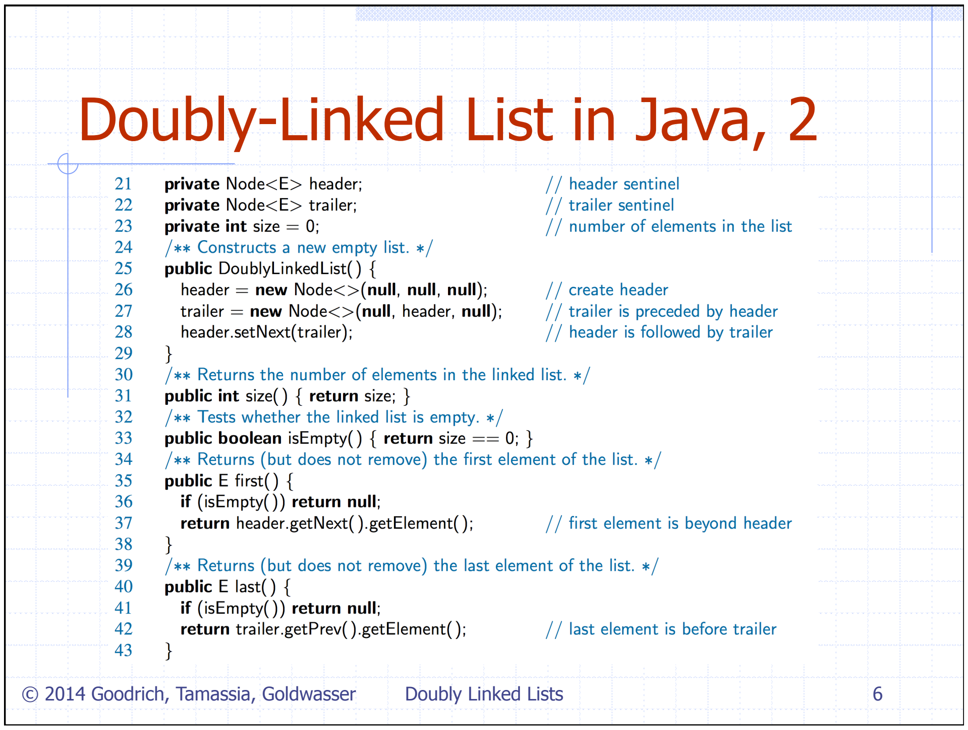
© 2014 Goodrich, Tamassia, Goldwasser Doubly Linked Lists 4



Doubly-Linked List in Java

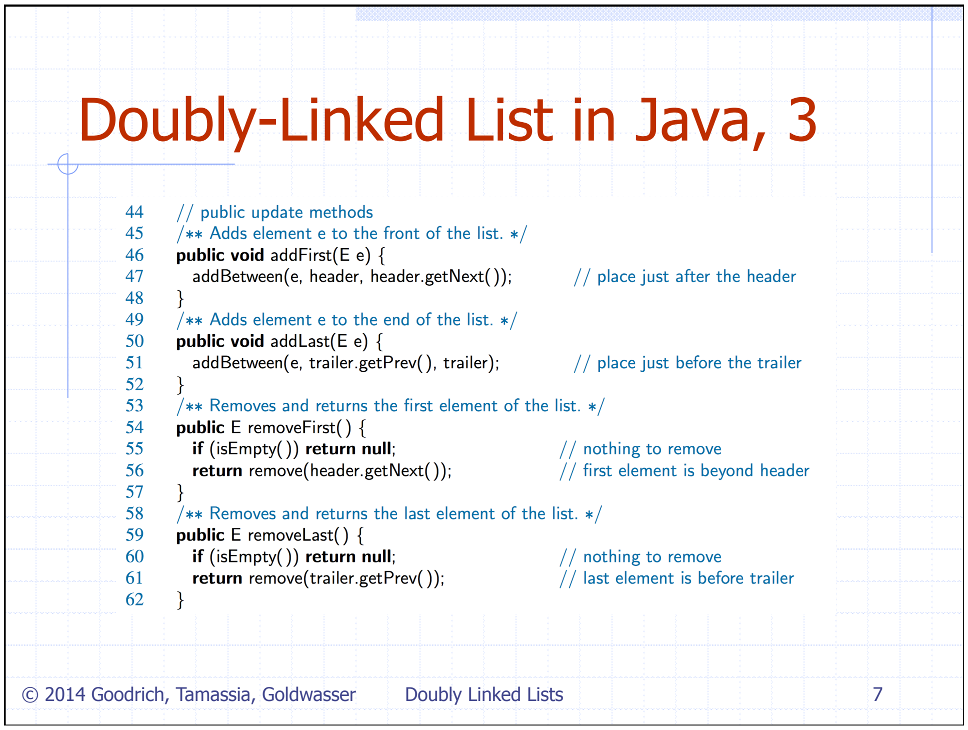
```
/** A basic doubly linked list implementation. */
public class DoublyLinkedList<E> {
  //---------------- nested Node class ----------------
  private static class Node<E> {
    private E element;              // reference to the element stored at this node
    private Node<E> prev;           // reference to the previous node in the list
    private Node<E> next;           // reference to the subsequent node in the list
    public Node(E e, Node<E> p, Node<E> n) {
      element = e;
      prev = p;
      next = n;
    }
    public E getElement( ) { return element; }
    public Node<E> getPrev( ) { return prev; }
    public Node<E> getNext( ) { return next; }
    public void setPrev(Node<E> p) { prev = p; }
    public void setNext(Node<E> n) { next = n; }
  } //----------- end of nested Node class -----------

```



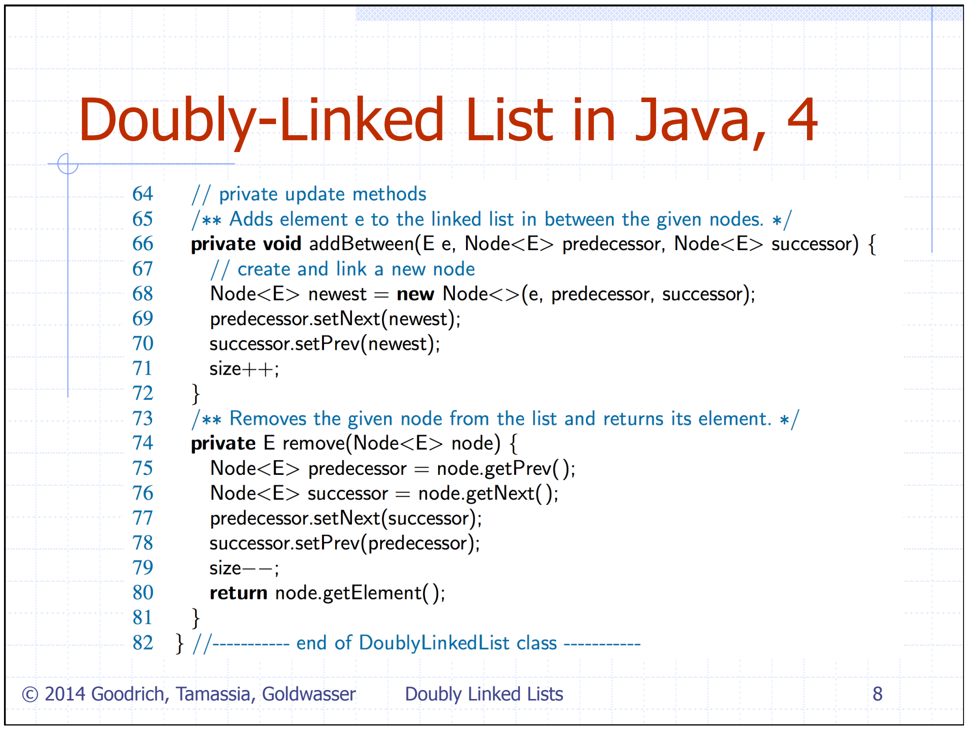
Doubly-Linked List in Java, 2

```
  private Node<E> header;                         // header sentinel
  private Node<E> trailer;                        // trailer sentinel
  private int size = 0;                           // number of elements in the list
  /** Constructs a new empty list. */
  public DoublyLinkedList( ) {
    header = new Node<>(null, null, null);        // create header
    trailer = new Node<>(null, header, null);     // trailer is preceded by header
    header.setNext(trailer);                      // header is followed by trailer
  }
  /** Returns the number of elements in the linked list. */
  public int size( ) { return size; }
  /** Tests whether the linked list is empty. */
  public boolean isEmpty( ) { return size == 0; }
  /** Returns (but does not remove) the first element of the list. */
  public E first( ) {
    if (isEmpty( )) return null;
    return header.getNext( ).getElement( );       // first element is beyond header
  }
  /** Returns (but does not remove) the last element of the list. */
  public E last( ) {
    if (isEmpty( )) return null;
    return trailer.getPrev( ).getElement( );      // last element is before trailer
  }
```



Doubly-Linked List in Java, 3

```
// public update methods
/** Adds element e to the front of the list. */
public void addFirst(E e) {
  addBetween(e, header, header.getNext( ));          // place just after the header
}
/** Adds element e to the end of the list. */
public void addLast(E e) {
  addBetween(e, trailer.getPrev( ), trailer);        // place just before the trailer
}
/** Removes and returns the first element of the list. */
public E removeFirst( ) {
  if (isEmpty( )) return null;                      // nothing to remove
  return remove(header.getNext( ));                 // first element is beyond header
}
/** Removes and returns the last element of the list. */
public E removeLast( ) {
  if (isEmpty( )) return null;                      // nothing to remove
  return remove(trailer.getPrev( ));                // last element is before trailer
}
```



Doubly-Linked List in Java, 4

```
  // private update methods
  /** Adds element e to the linked list in between the given nodes. */
  private void addBetween(E e, Node<E> predecessor, Node<E> successor) {
    // create and link a new node
    Node<E> newest = new Node<>(e, predecessor, successor);
    predecessor.setNext(newest);
    successor.setPrev(newest);
    size++;
  }
  /** Removes the given node from the list and returns its element. */
  private E remove(Node<E> node) {
    Node<E> predecessor = node.getPrev( );
    Node<E> successor = node.getNext( );
    predecessor.setNext(successor);
    successor.setPrev(predecessor);
    size--;
    return node.getElement( );
  }
} //----------- end of DoublyLinkedList class -----------
```