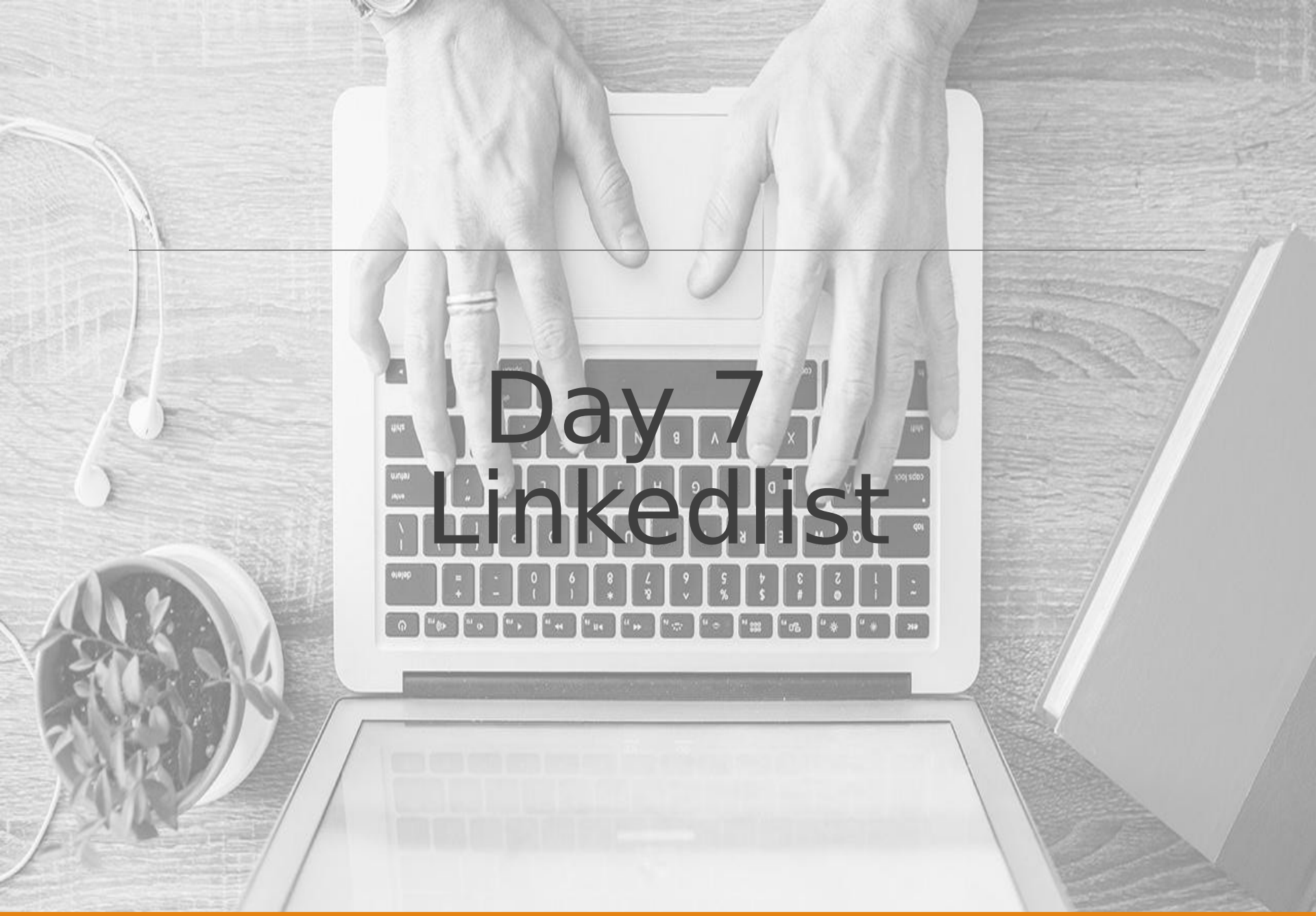


Day 7
Linkedlist

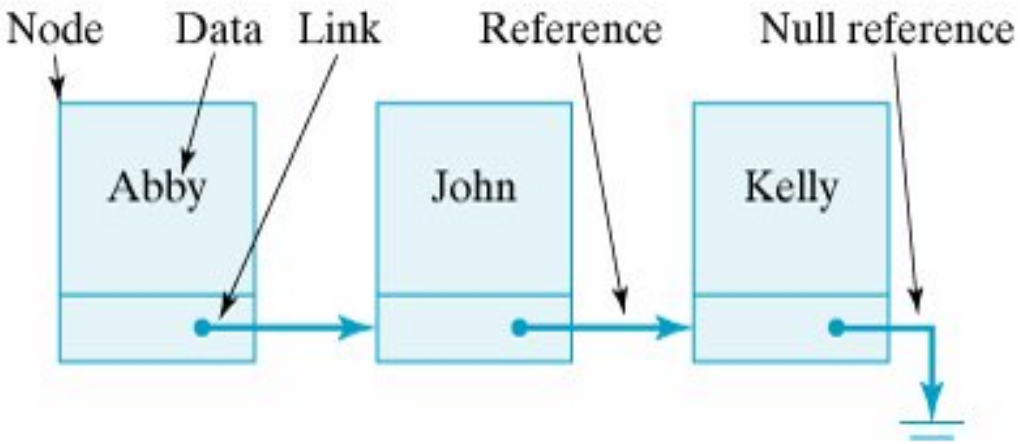
Agenda

- Introduce Linkedlist
- Creating a Linkedlist
- Insert a Linkedlist
- Traverse Linkedlist
- Remove Linkedlist

THE LINKED LIST

- A **linked list** is a list in which a collection of nodes are linked together by references from one node to the next. To make a linked list, we will define a class of self-referential objects. A **self-referential object** is an object that contains a reference to an object of the same class

Figure 16.1. A linked list of `Node`s terminated by a `null` link.



Self-Referential Object : object is one that contains an instance variable that refers to an object of the same

NODE EXAMPLE

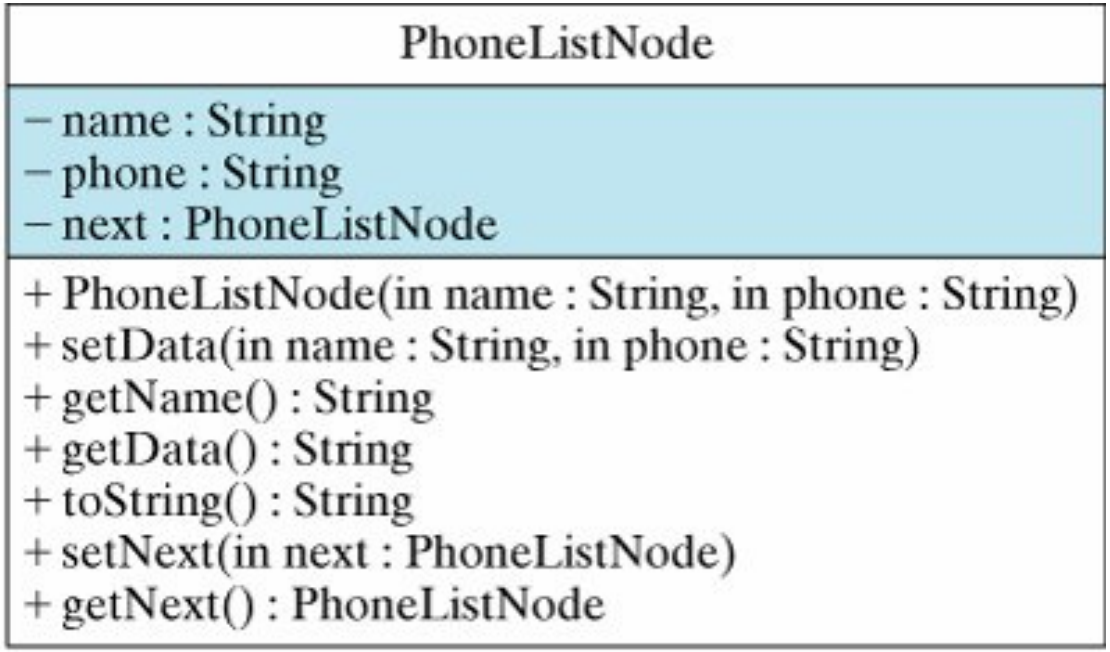
```
public class Node {
    private Object data;
    private Node next;

    public Node(Object obj);          // Constructor

    public void setData(Object obj);  // Data access
    public Object getData();

    public void setNext(Node link);   // Link access
    public Node getNext();
} // Node class
```

EXAMPLE: THE DYNAMIC PHONE LIST



THE DYNAMIC PHONE LIST

```
public class PhoneListNode {
     private String name;
     private String phone;
     private PhoneListNode next;

     public PhoneListNode(String s1, String s2) {
          name = s1;
          phone = s2;
          next = null;
     } // PhoneListNode()

     public void setData(String s1, String s2) {
          name = s1;
          phone = s2;
     } // setData()

     public String getName() {
          return name;
     } // getName()

     public String getData() {
          return name + " " + phone;
     } // getData()

     public String toString() {
          return name + " " + phone;
     } // toString()

     public void setNext(PhoneListNode nextPtr) {
          next = nextPtr;
     } // setNext()

     public PhoneListNode getNext() {
          return next;
     } // getNext()
} // PhoneListNode class
```

Note that the constructor sets the initial value of next to null, which means that it refers to no object

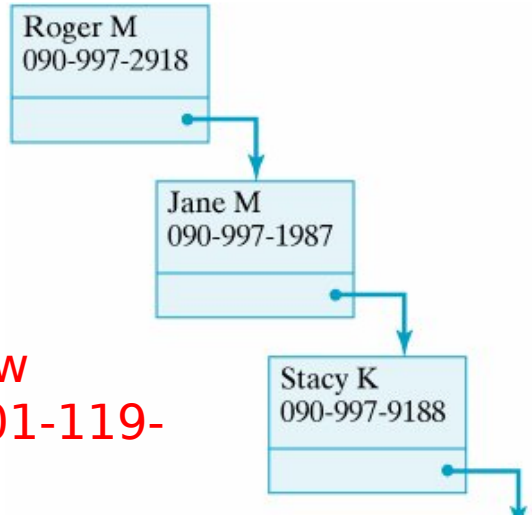
PHONE LIST

```
PhoneListNode node1 = new PhoneListNode("Roger M","090-997-2918");
PhoneListNode node2 = new PhoneListNode("Jane M","090-997-1987");
PhoneListNode node3 = new PhoneListNode("Stacy K","090-997-9188");
```

The next two statements chain the nodes together into the list shown in Figure 16.5:

```
node1.setNext(node2);
node2.setNext(node3);
```

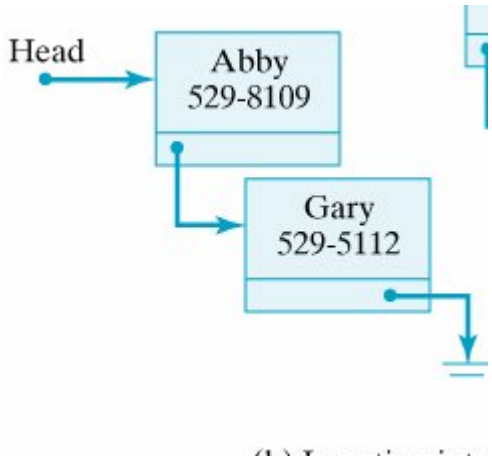
Figure 16.5. The phone list: a linked list of nodes, each of which contains a person's name and phone number.




```
PhoneListNode node4 = new PhoneListNode("gary g","201-119-8765");
node3.setNext(node4);
```

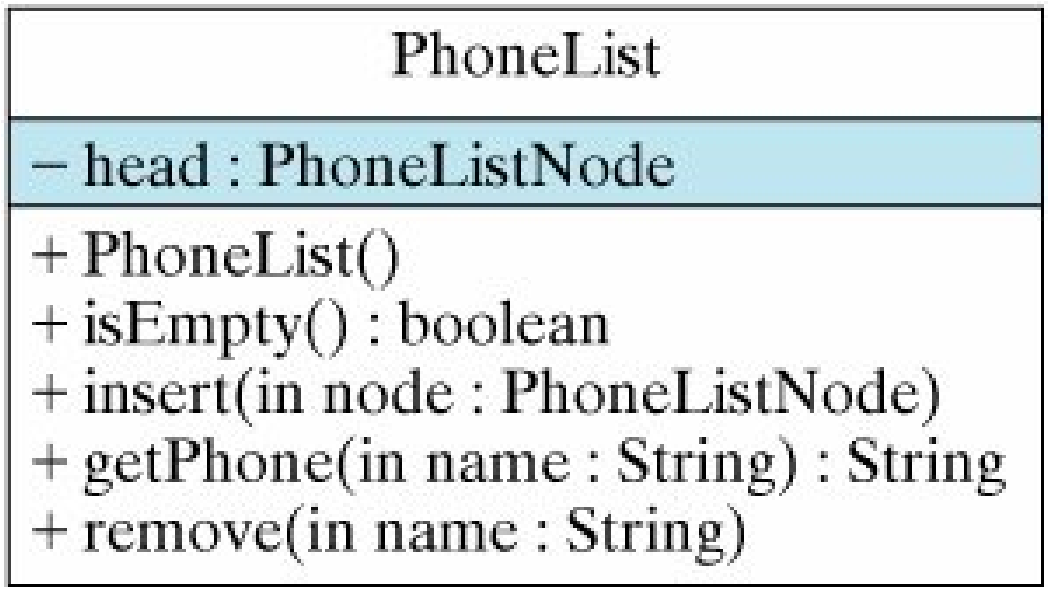
MANIPULATING THE PHONE LIST

- This class will include the insert, access, and remove methods. It must also contain a reference to the list itself.
- Head



THE PHONE LIST INTERFACE

- We will implement these feature below for phone list



IMPLEMENTATION PHONELIST

```
public class PhoneList {
    private PhoneListNode head;

    public PhoneList() {
        head = null;
    }
    public boolean isEmpty() {
        return head == null;
    }
    public void insert(PhoneListNode node) { }
    public String getPhone(String name) { }
    public String remove(String name) { }
    public void print() { }
} // PhoneList class
```

when a new PhoneList instance is constructed, head is initialized to null, meaning that the list is initially emptywhen a new PhoneList instance is constructed, head is initialized to null, meaning that the list is initially empty

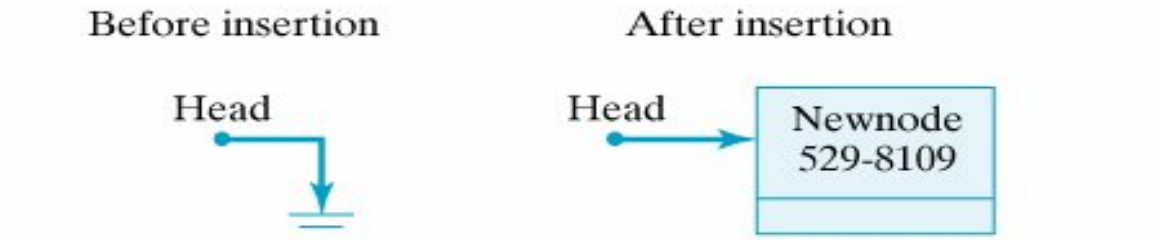
want to test whether the list is empty, we define the boolean isEmpty() method for that purpowant to test whether the list is empty, we define the boolean isEmpty() method for that purpo

INSERTING NODES INTO A LIST

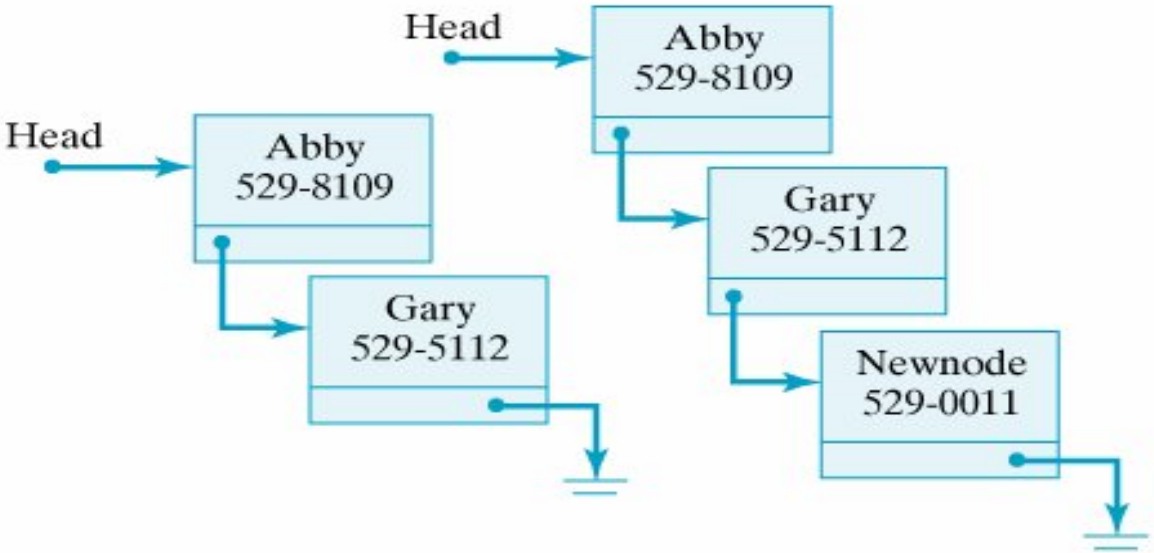
- The node could be inserted at the beginning or end of the list, or in alphabetical order, or possibly in other ways

- There are two cases we need to worry about for this algorithm
  - if the list is empty, we can insert the node by simply setting head to point to the node
  - if the list is not empty, we must move through, or traverse, the links of the list until we find the last node and insert the new node

INSERTING NODES INTO A LIST



(a) Insertion into empty list

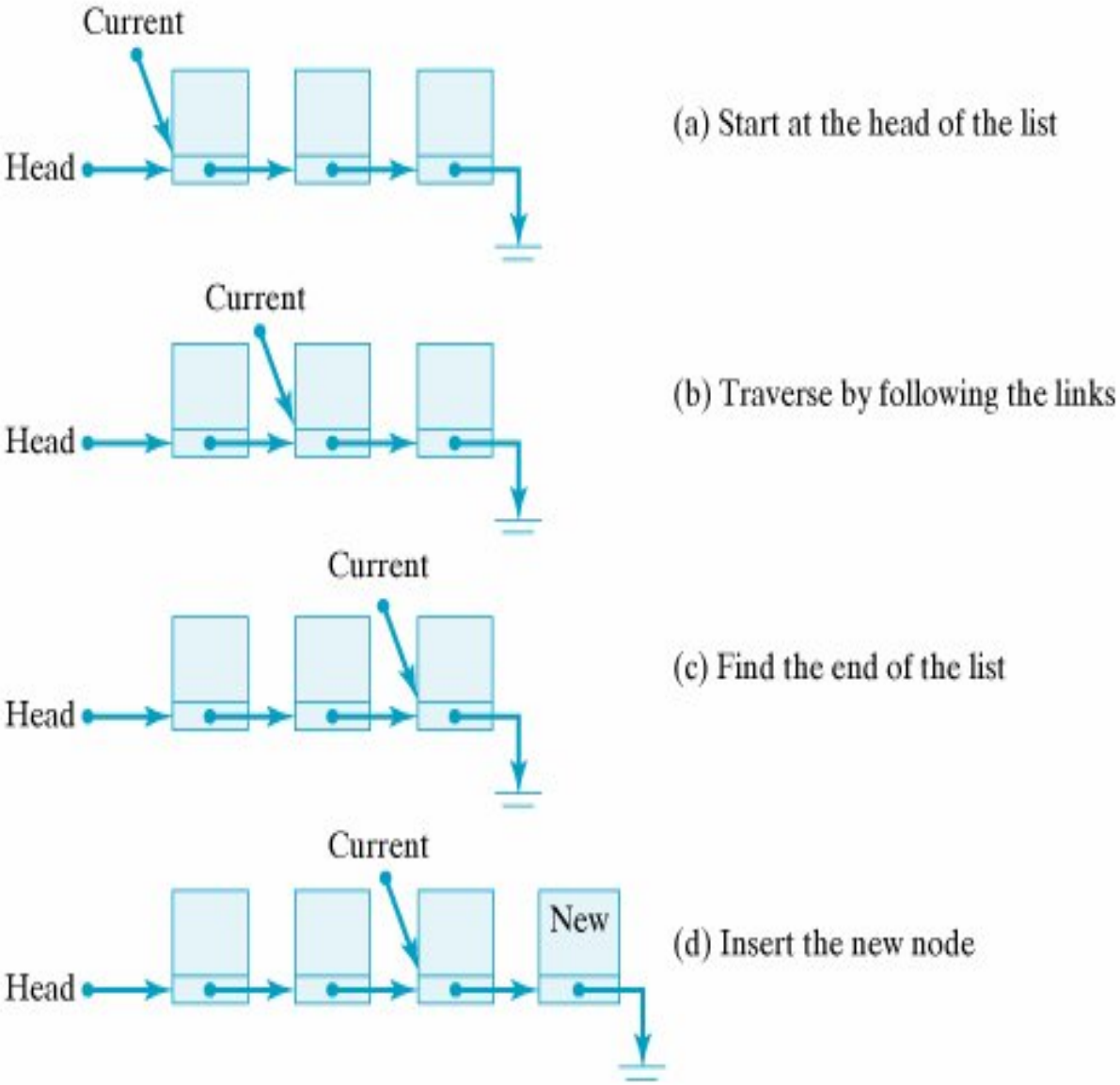


(b) Insertion into existing list

ALGORITHMS

```
public void insert(PhoneListNode newNode) {
   if (isEmpty())
     head = newNode;                          // Insert at head of list
   else {
     PhoneListNode current = head;             // Start traversal at head
     while (current.getNext() != null)         // While not last node
       current = current.getNext();            // go to next node
       current.setNext( newNode );             // Do the insertion
     }
} // insert()
```

Figure 16.9. The temporary variable `current` TRaverses the list to find its end.



PRINTING THE NODES OF A LIST

```
public void print() {
   if (isEmpty())
      System.out.println("Phone list is empty");
   PhoneListNode current = head;                    // Start traversal at head
   while (current != null) {                        // While not end of list
     System.out.println( current.toString() );      // print data
     current = current.getNext();                   // go to next node
   }
} // print()
```

LOOKING UP A NODE IN A LIST

- we start at the head of the list and traverse through the next links until we find the node containing the desired phone number. This method takes the name of the person as a parameter

```
public String getPhone(String name) {
   if (isEmpty())                                          // Case 1: Empty list
     return "Phone list is empty";
   else {
     PhoneListNode current = head;
     while ((current.getNext() != null) && (!current.getName().equals(name)))
       current = current.getNext();
       if (current.getName().equals(name))
         return current.getData();                         // Case 2: Found name
       else                                                // Case 3: No such person
         return ("Sorry.   No entry for " + name);
    }
} // getPhone()
```

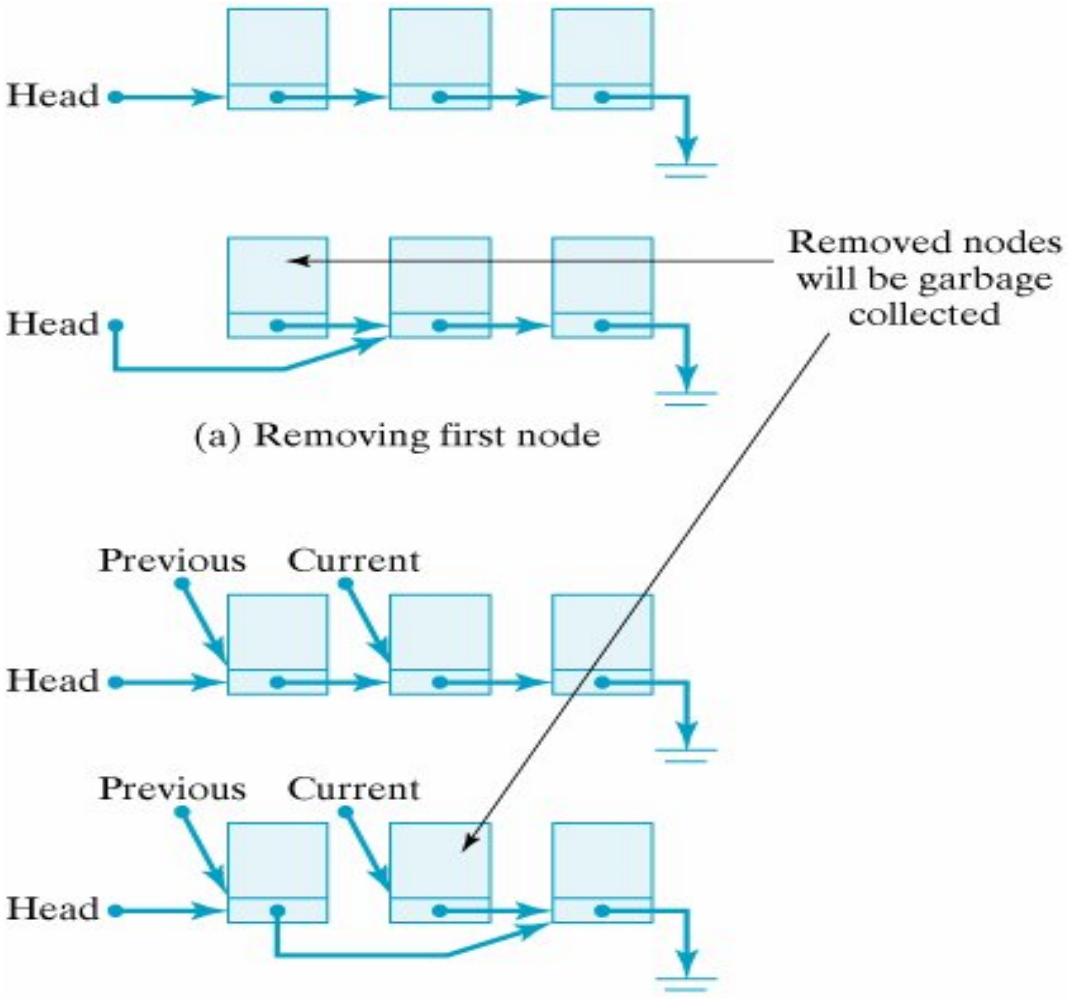
REMOVE NODE

- If the list is empty, we just return an error message
- We use current as the traversal variable
- If the named node is the first node, we simply need to set head to current.getNext(), which has the effect of making head point to the second node in the list -> Once the node is cut out from the chain of links, there will be no further reference to it. In this case, Java will recapture the memory it uses when it does garbage collection

REMOVE NODE

```
public String remove(String name) {                    // Remove an entry by name
   if (isEmpty())                                      // Case 1: empty list
     return "Phone list is empty";
   PhoneListNode current = head;
   PhoneListNode previous = null;
   if (current.getName().equals(name)) {               // Case 2: remove first node
     head = current.getNext();
     return "Removed " + current.toString() ;
   }
   while ((current.getNext() != null) && (!current.getName().equals(name))) {
     previous = current;
     current = current.getNext();
   }
   if (current.getName().equals(name)) {               // Case 3: remove named node
     previous.setNext(current.getNext());
     return "Removed " + current.toString();
   } else
     return ("Sorry. No entry for " + name);           // Case 4: node not found
} // remove()
```

Figure 16.11. Removing different nodes from a linked list.



EXCISE

Write a statement to create a new Node whose data element consists of the Student named "William". Assume that the Student class has a constructor with a String parameter for the student's name

Thanks!

Any questions?