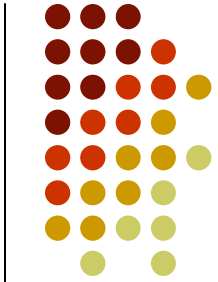
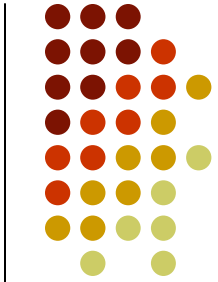


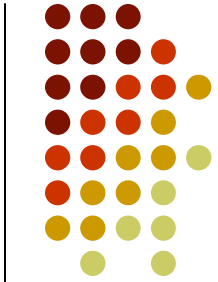
Linked Lists

Preliminaries

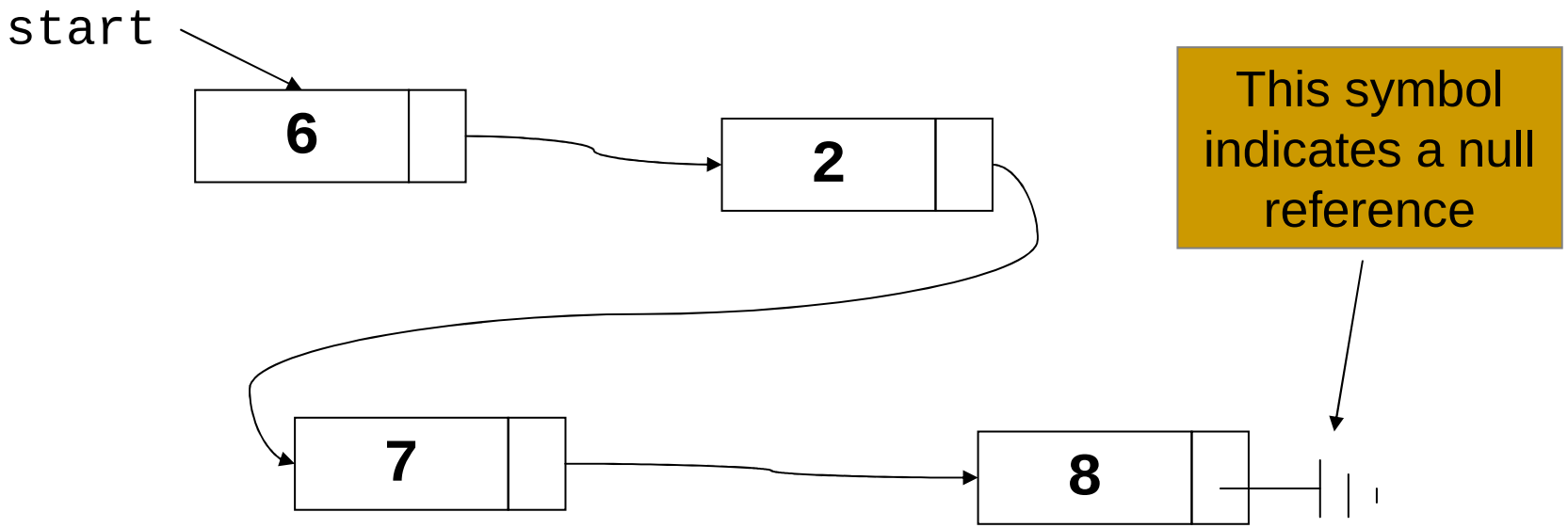


- Options for implementing an ADT List
  - Array
    - Has a fixed size
    - Data must be shifted during insertions and deletions
  - Dynamic array
  - Linked list
    - Is able to grow in size as needed
    - Does not require the shifting of items during insertions and deletions
    - Accessing an element does not take a constant time

Linked Lists

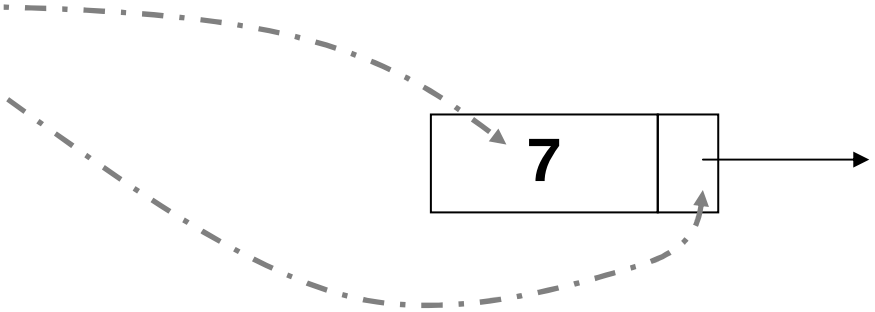


- A linked list is a dynamic data structure consisting of nodes and links:



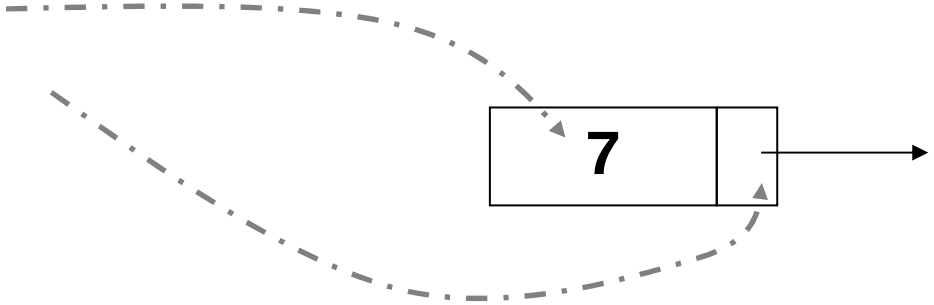
Linked Lists

- Each node contains two things
  - The data
  - A pointer to the next node



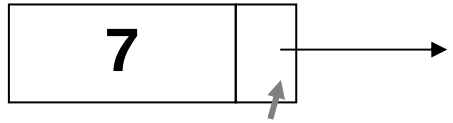
Linked Lists

```
class Node {
    public int data;
    public Node next;
}
```



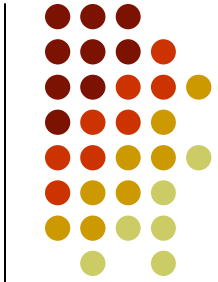
Linked Lists

```
class Node {
    public int data;
    public Node next;
}
```



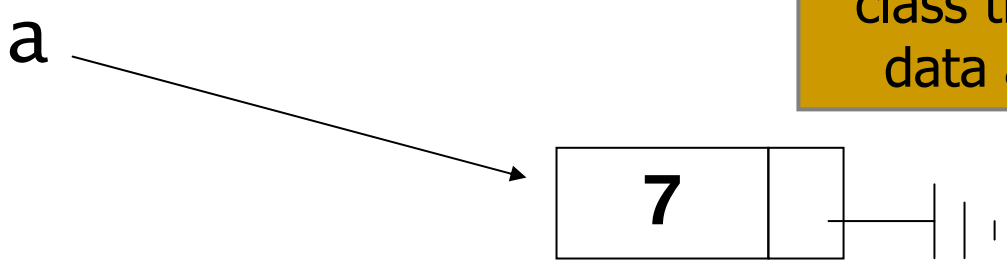
A node points to another node, so the pointer must be of type Node

Building a Linked List



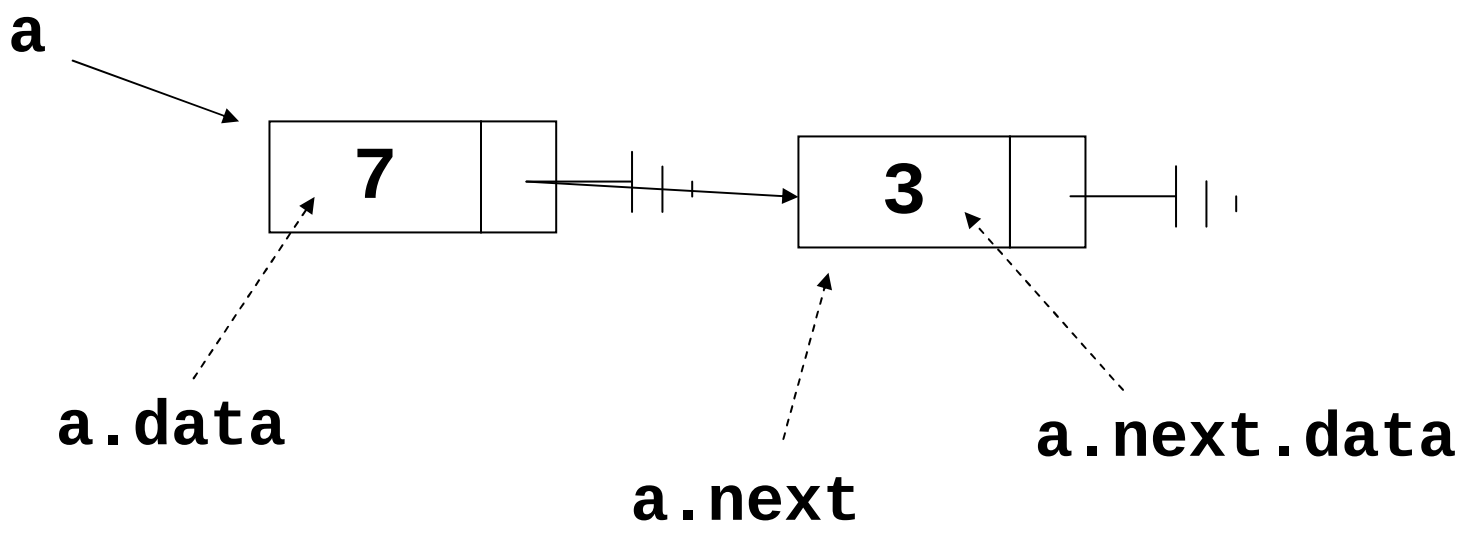
```
Node a = new Node(7, null);
```

Assume we've added a constructor to the Node class that lets us initialize data and next like this.

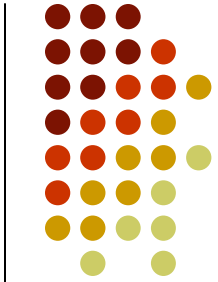


Building a Linked List

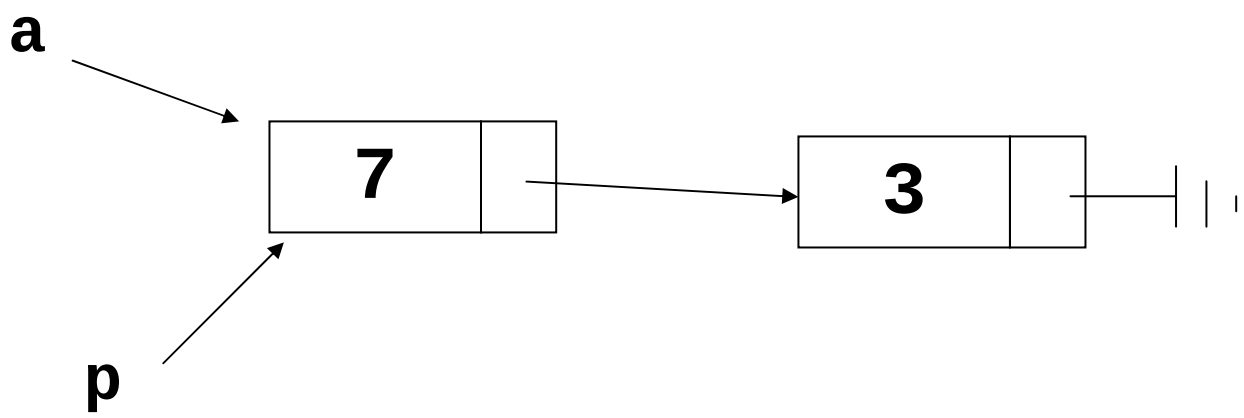
```
Node a = new Node(7, null);
a.next = new Node(3, null);
```



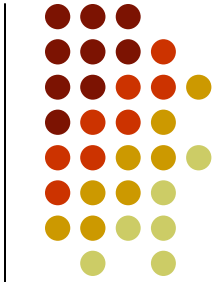
Traversing a Linked List



```
Node a = new Node(7, null);
a.next = new Node(3, null);
Node p = a;
```

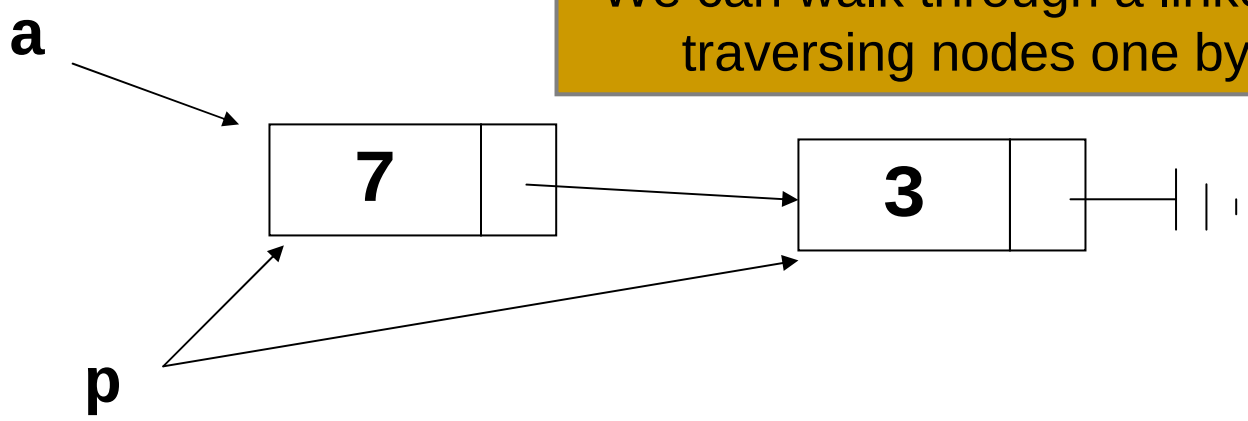


Building a Linked List

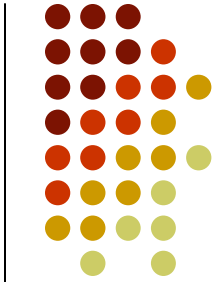


```
Node a = new Node(7, null);
a.next = new Node(3, null);
Node p = a;
p = p.next;  // go to the next node
```

We can walk through a linked list by traversing nodes one by one.

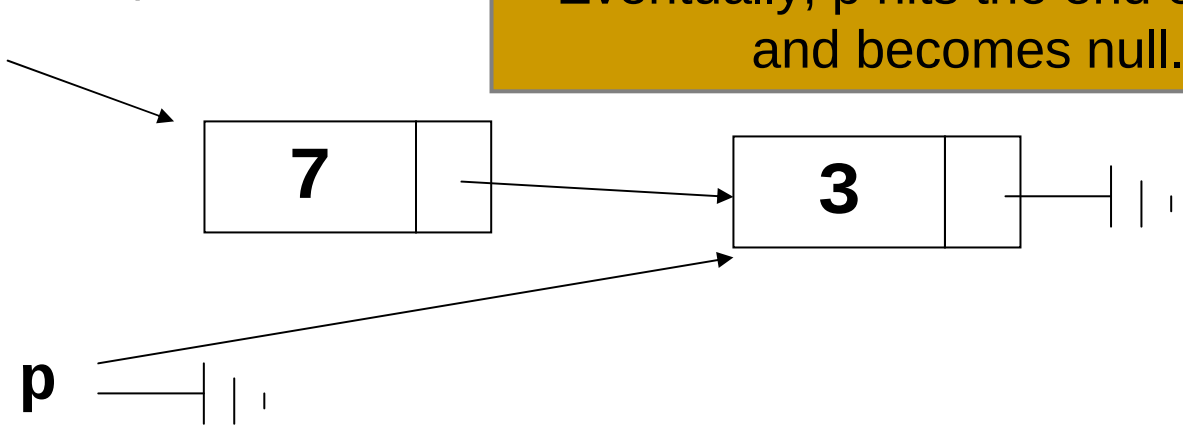


Traversing a Linked List



```
Node a = new Node(7, null);
a.next = new Node(3, null);
Node p = a;
p = p.next;  // go to the next node
p = p.next;
```

Eventually, p hits the end of the list and becomes null.



Preliminaries

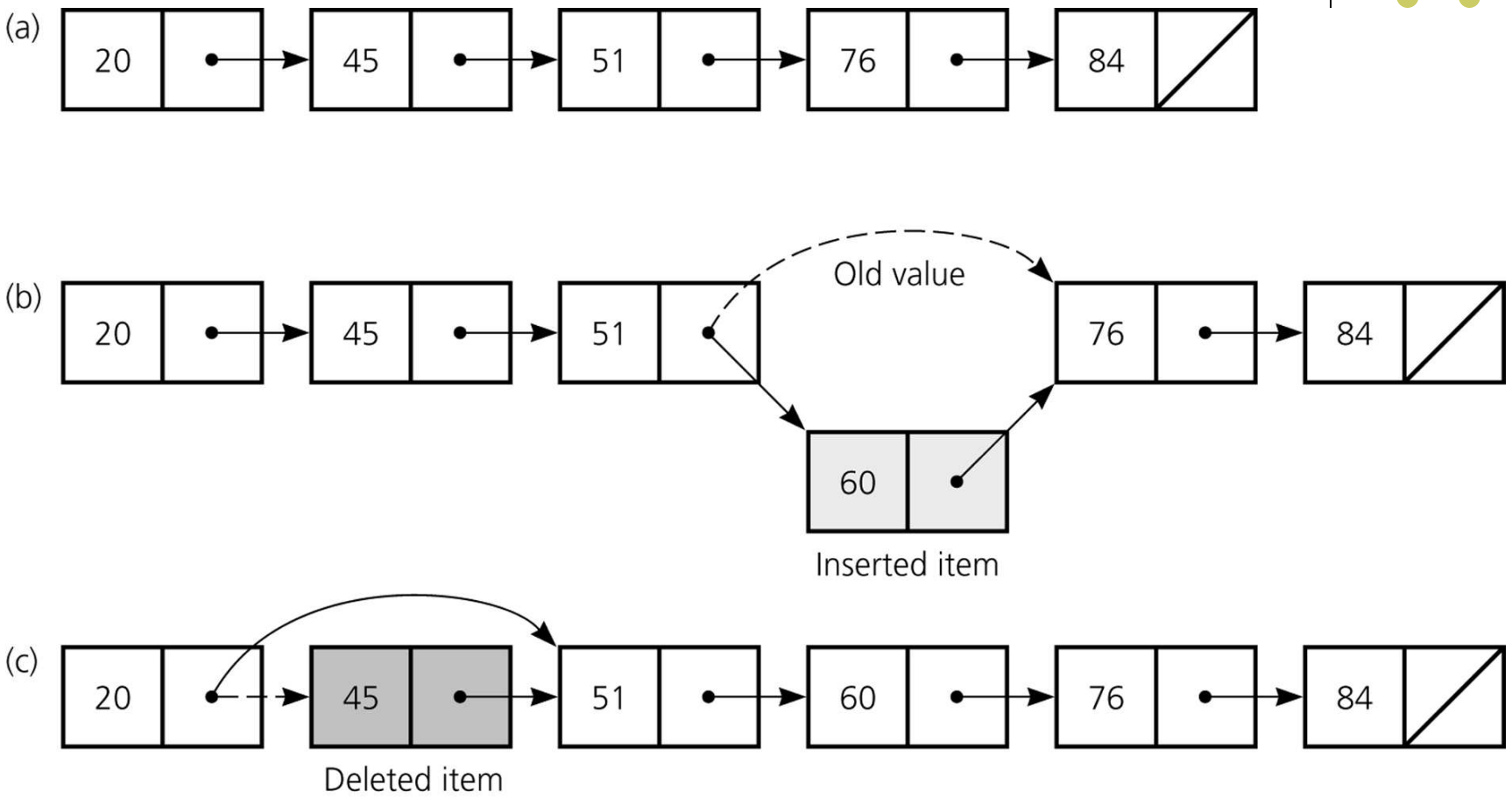
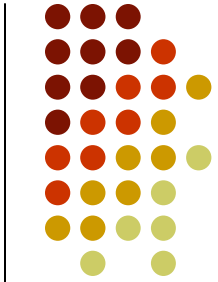


Figure 5-1

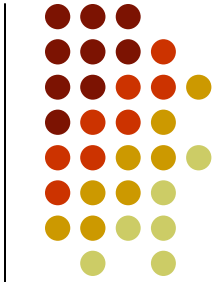
a) A linked list of integers; b) insertion; c) deletion

Reference-based implementation



- Reference-based implementations of ADT use Java references
- We are going to look at a reference-based implementation of ATD list (using linked lists)

Object References (review)

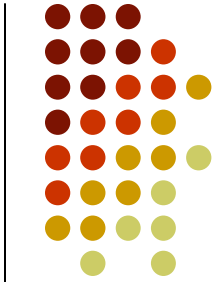


- A reference variable
  - Contains the location of an object
  - Example

```
Node node;
node = new Node(5,null);
```

  - As a data field of a class
    - has the default value `null`
  - As a local reference variable (in a method)
    - does not have a default value

Object References



- When one reference variable is assigned to another reference variable, both references then refer to the same object

```
Integer p, q;
p = new Integer(6);
q = p;
```

- A reference variable that no longer references any object is marked for garbage collection

Object References

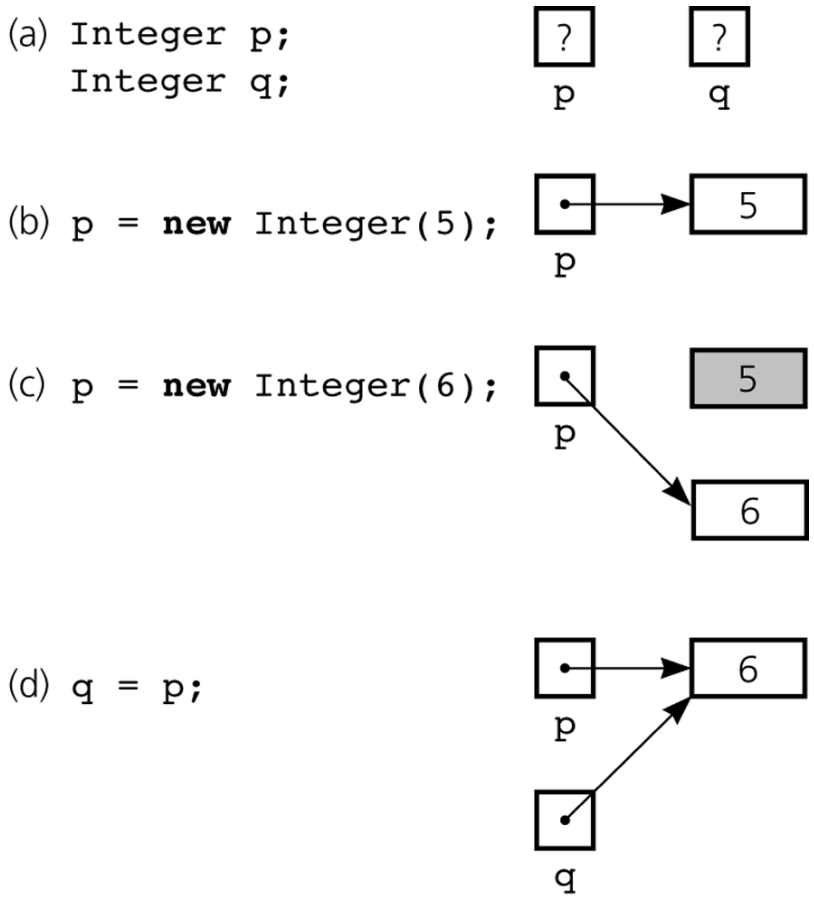


Figure 5-3a-d

a) Declaring reference variables; b) allocating an object; c) allocating another object, with the dereferenced object marked for garbage collection

Object References

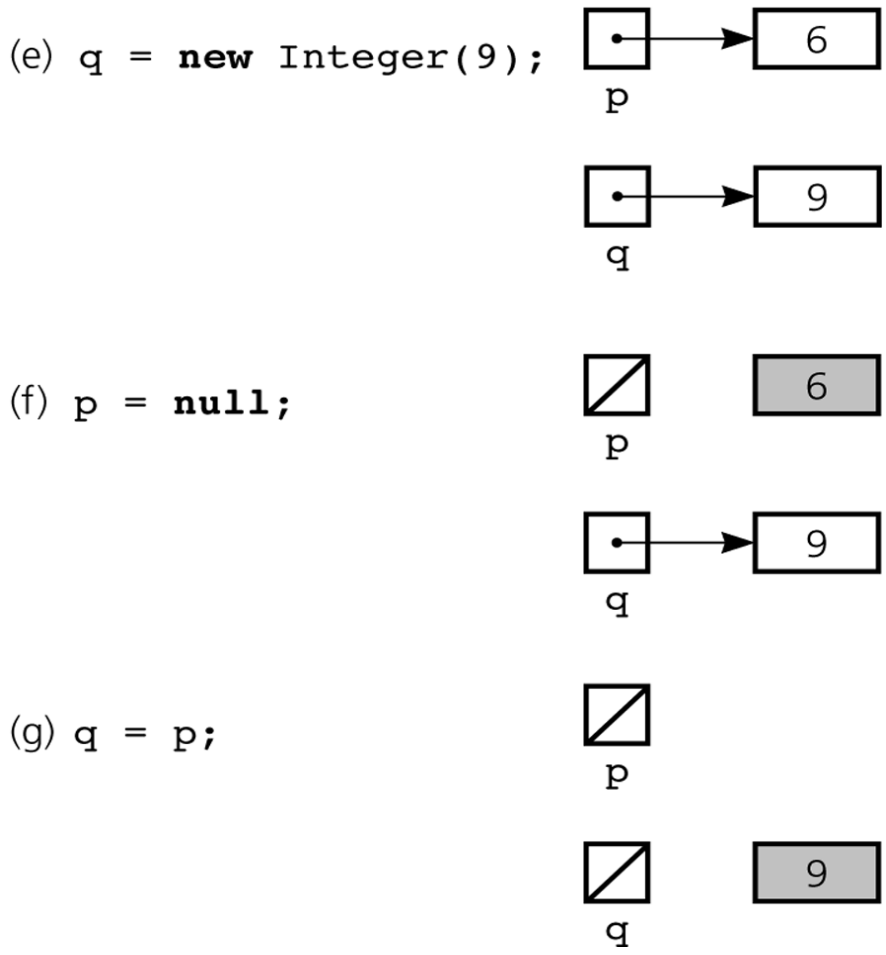
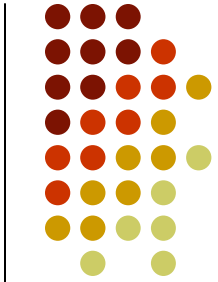


Figure 5-3e-g

e) allocating an object; f) assigning ***null*** to a reference variable;

g) assigning a reference with a ***null*** value

Object References (arrays)



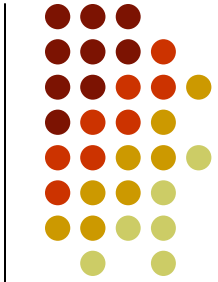
- An array of objects
  - Is actually an array of references to the objects
  - Example

```
Integer[] scores = new Integer[30];
```

  - Instantiating Integer objects for each array reference

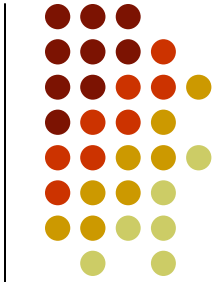
```
scores[0] = new Integer(7);
scores[1] = new Integer(9); // and so on …
```

Object References (equality)



- Equality operators (== and `!=`)
  - Compare the values of the reference variables, not the objects that they reference
- `equals` method
  - Compares objects by contents

Object References



- When an object is passed to a method as an argument, the reference to the object is copied to the method's formal parameter

```
public void changeInteger(Integer n)
{
  n=new Integer(5);
}
Integer p=new Integer(7);
changeInteger(p);
```

Reference-Based Linked Lists

- Linked list
  - Contains nodes that are linked to one another
  - A node
    - Contains both data and a “link” to the next item
    - Can be implemented as an object

```
public class Node {
     private Object item;
     private Node next;
     // constructors, accessors,
     // and mutators …
} // end class Node
```

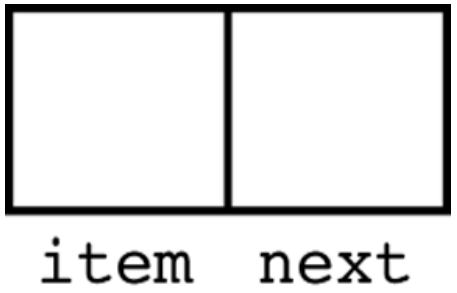


Figure 5-5

A node

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

  public Node(Object newItem) {
    item = newItem;
    next = null;
  } // end constructor

  public Node(Object newItem, Node nextNode) {
    item = newItem;
    next = nextNode;
  } // end constructor

  public void setItem(Object newItem) {
    item = newItem;
  } // end setItem

  public Object getItem() {
    return item;
  } // end getItem

  public void setNext(Node nextNode) {
    next = nextNode;
  } // end setNext

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

Reference-Based Linked Lists

- Using the Node class

```
Node n = new Node (new Integer(6));
Node first = new Node (new Integer(9), n);
```

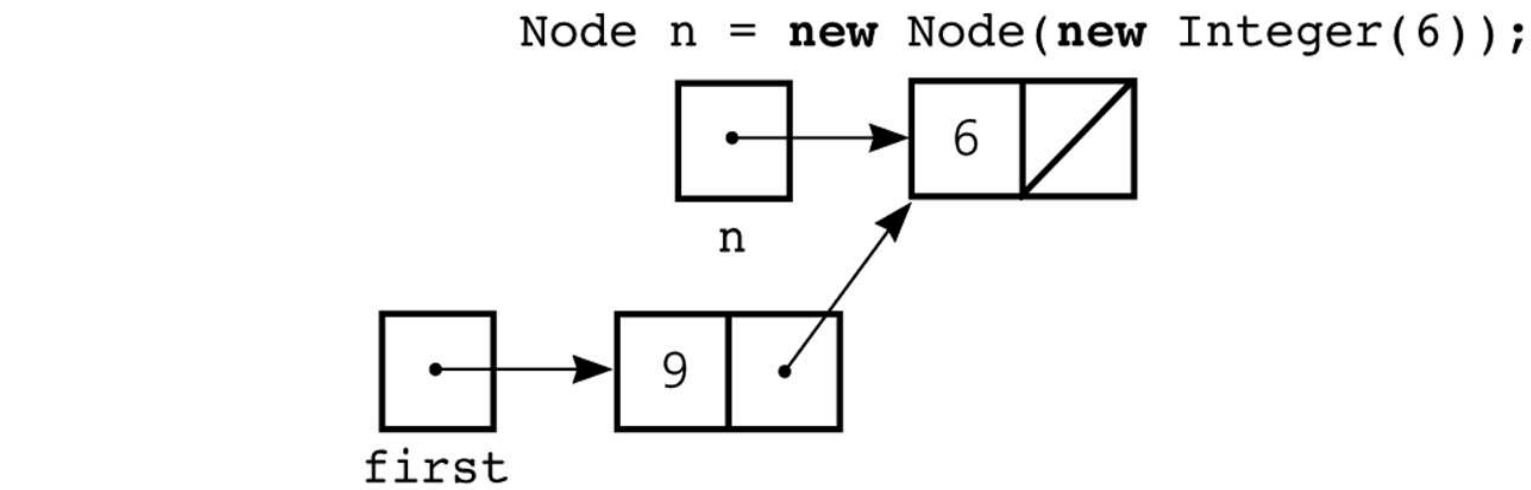


Figure 5-7

Using the ***Node*** constructor to initialize a data field and a link value

Reference-Based Linked Lists

- Data field `next` in the last node is set to `null`
- `head` reference variable
  - References the list's first node
  - Always exists even when the list is empty

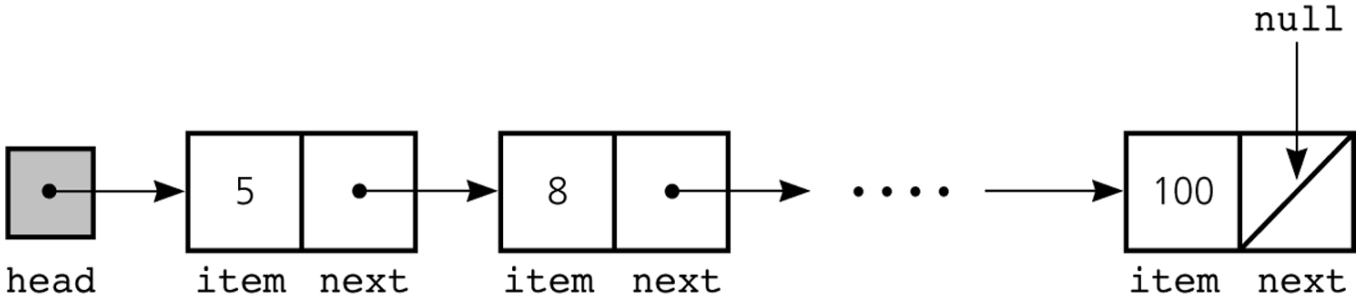
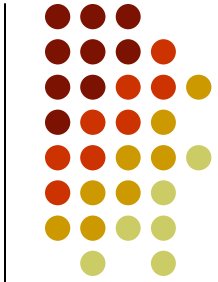


Figure 5-8

A ***head*** reference to a linked list

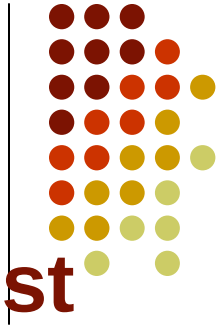
Reference-Based Linked Lists



```
public class ListReferenceBased
  implements ListInterface {
  // reference to linked list of items
  private Node head;
  private int numItems;
  // number of items in list

  public ListReferenceBased() {
    numItems = 0;
    head = null;
  }  // end default constructor
```

Programming with Linked Lists: Displaying the Contents of a Linked List



- `curr` reference variable
  - References the current node
  - Initially references the first node
- To display the data portion of the current node

```
System.out.println(curr.getItem());
```

- To advance the current position to the next node

```
curr = curr.getNext();
```

Displaying the Contents of a Linked List

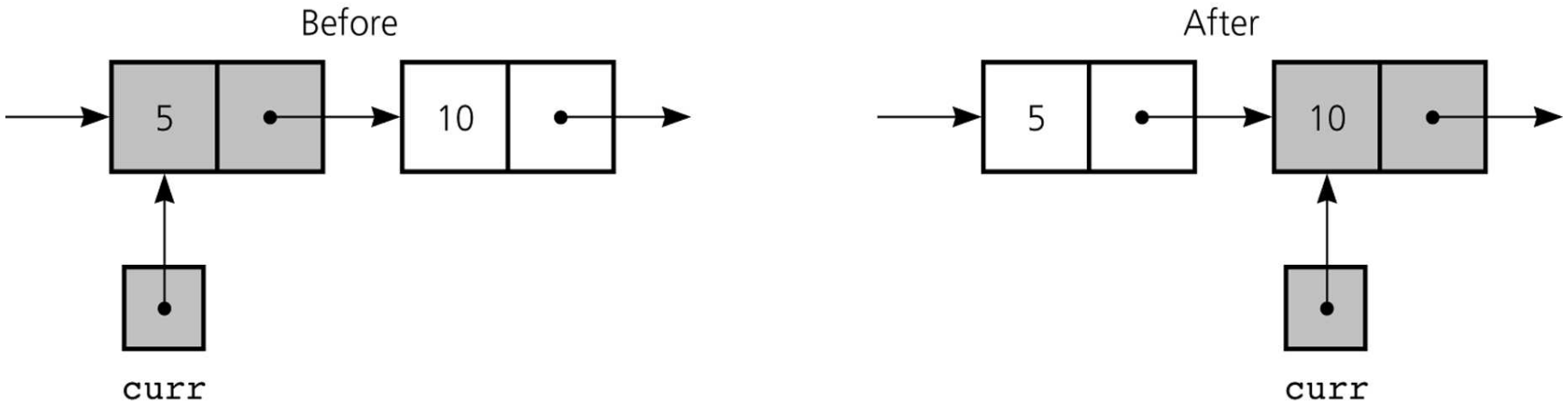


Figure 5-10

The effect of the assignment ***curr = curr.getNext( )***

Displaying the Contents of a Linked List

- To display all the data items in a linked list

```
for (Node curr = head; curr != null; curr =
     curr.getNext()) {
  System.out.println(curr.getItem());
} // end for
```

Deleting a Specified Node from a Linked List

- To delete node N which `curr` references
  - Set `next` in the node that precedes N to reference the node that follows N

```
prev.setNext(curr.getNext());
```

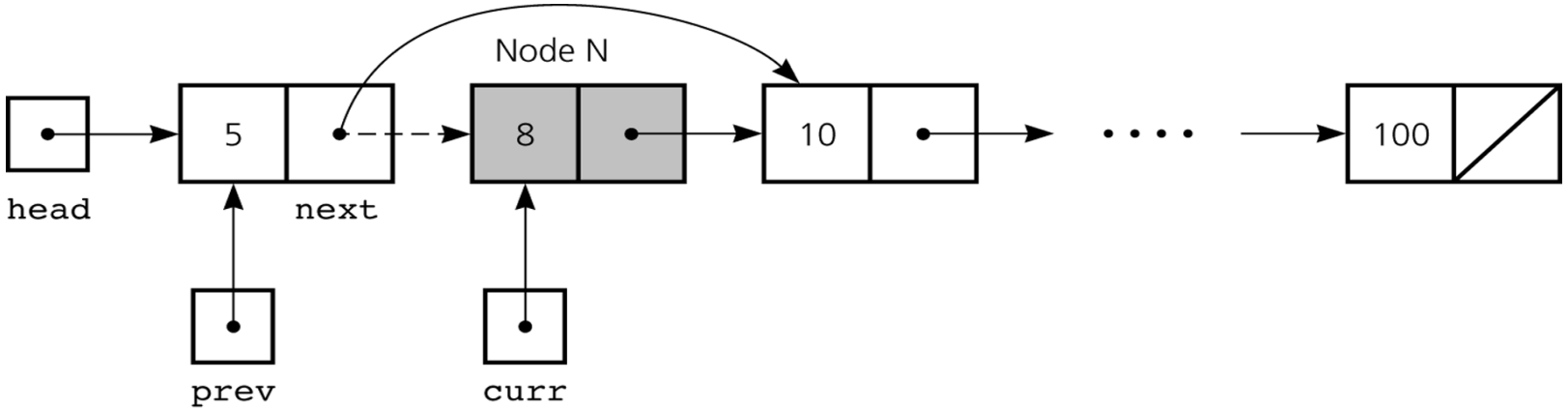


Figure 5-11

Deleting a node from a linked list

Deleting a Specified Node from a Linked List

- Deleting the first node is a special case

```
head = head.getNext();
```

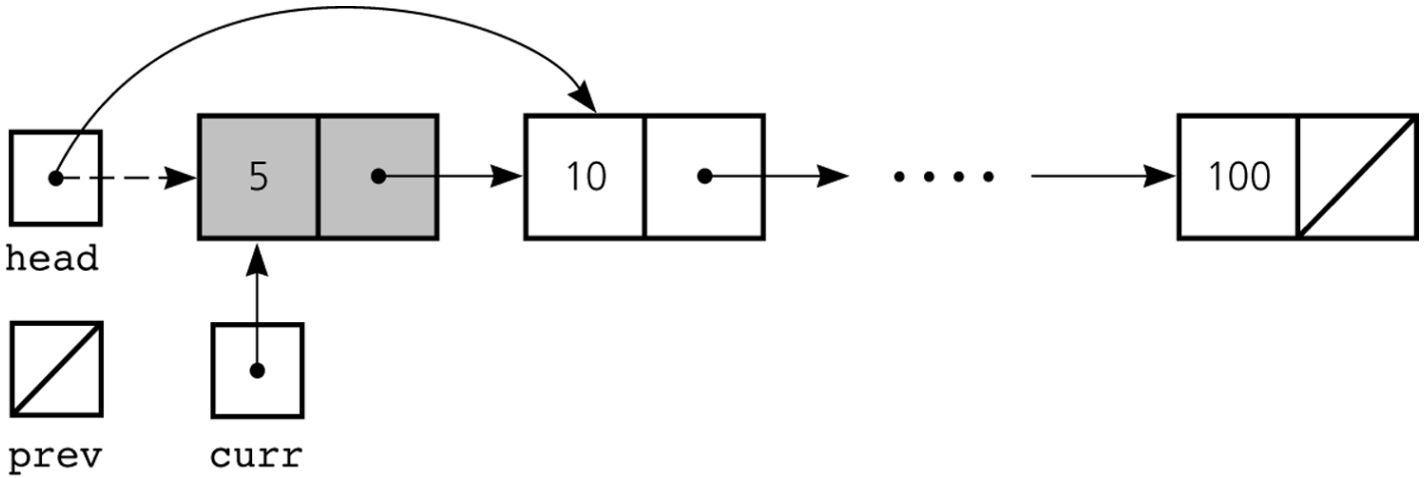


Figure 5-12

Deleting the first node

Deleting a Specified Node from a Linked List

- To return a node that is no longer needed to the system

```
curr.setNext(null);
curr = null;
```

- Three steps to delete a node from a linked list
  - Locate the node that you want to delete
  - Disconnect this node from the linked list by changing references
  - (Return the node to the system)