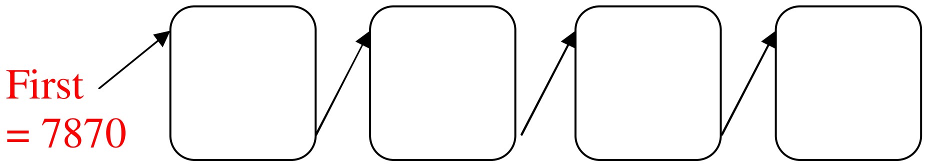
Bags – Part II

- References:
  - Text book : Chapter 3
  - Oracle/Sun Java Tutorial : http://java.sun.com/j2se/1.5.0/docs/guide/collections/index.html

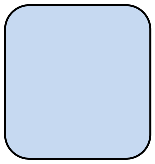
1. Bag Implementation That Links Data (Linked List)

- Strategy: A list of nodes connect (or link) together. Only know ID of first node. Example:

ID	7870	9092	8989	6543	
Next	9092	8989	6543	--	
OID	122	564	323	222	// data object

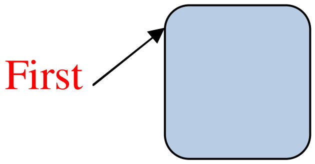


- Add a new node to the beginning NewNode

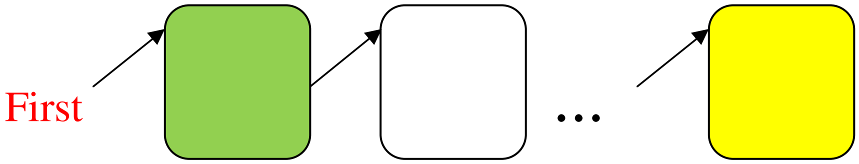


Two cases:

Case 1: The list is empty. First = NewNode

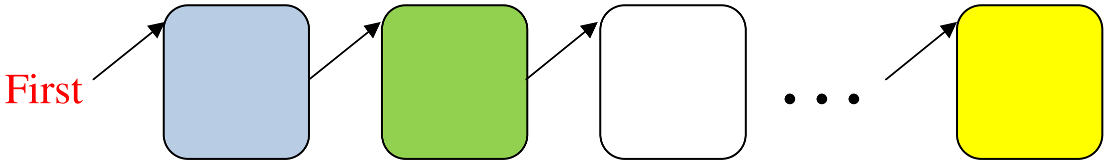


Case 2: The list is not empty, i.e. one or more



Steps:

```
NewNode.Next = First   // let new node remember next node
First = NewNode        // update First to remember new node
```

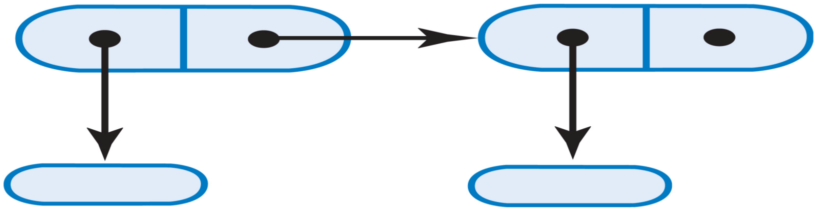


- The class Node

  - Nodes are objects that are linked together to form a data structure
  - We will use nodes with two data fields
  - A reference to an entry in the list // OID or Data
  - A reference to another node // Next

```
private class Node
{
   private T data; // entry in list
   private Node next; // link to next node

   private Node (T dataPortion)
   {
        data = dataPortion;
        next = null;    // set next to NULL
   } // end constructor


   private Node (T dataPortion, Node nextNode)
   {
        data = dataPortion;
        next = nextNode;  // set next to refer to nextNode
   } // end constructor

} // end Node
```

Two nodes linked together (data is referenced to entries)



- A Linked List Implementation of the ADT Bag
    - Use a chain of nodes
    - Remember the address of the first node in the chain
      i.e. Record a reference to the first node.

- Contains the class **Node** as a private inner class. Its data fields can be accessed by enclosing class without the need for accessor or mutator methods.

- For empty bag, firstNode=NULL
  For non-empty bag, last node next is always NULL

- To implement a class, you should implement a few core methods, then test those methods before implement more methods

  For bag

  - Need to be able to create the collection: constructor and methods add()

    Also implement simple methods: isEmpty(), clear() etc

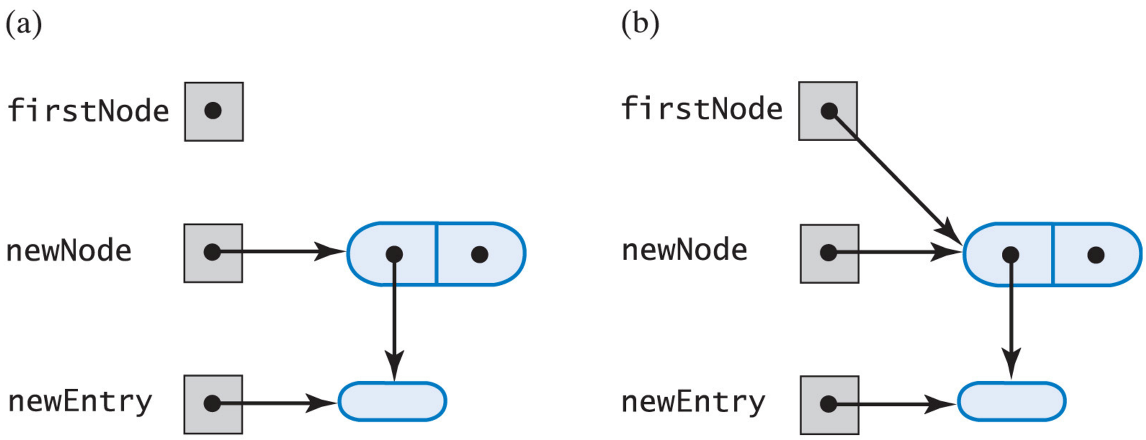
  - Explain program in next few slides

```
public class LinkedBag < T > implements BagInterface < T >
{
   private Node firstNode; // reference to first node
   private int numberOfEntries;     // number of entries in list

    public LinkedBag ()
    {
       firstNode = null;
       numberOfEntries = 0;
     } // end default constructor

     // Implementations of the public methods declared in BagInterface….

     private class Node // private inner class
     {
       // as given previously
     } // end Node
}
```

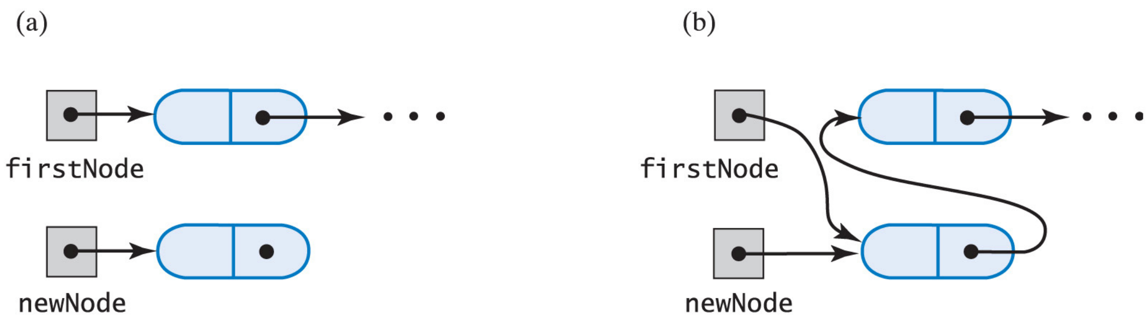
- Add : easy to add to the beginning

```
public boolean add (T newEntry)
{
   // get new node to hold newEntry
   // system out-of-memory error may happen,
   // usually, not much you can do
   Node newNode = new Node (newEntry);
   // case I
   if (isEmpty ())
      firstNode = newNode;
   else // Case II: add to beginning of nonempty list
   {
      newNode.next = firstNode; // make new node reference first node
      firstNode = newNode;
   } // end if

   numberOfEntries++;
   return true;
} // end add
```

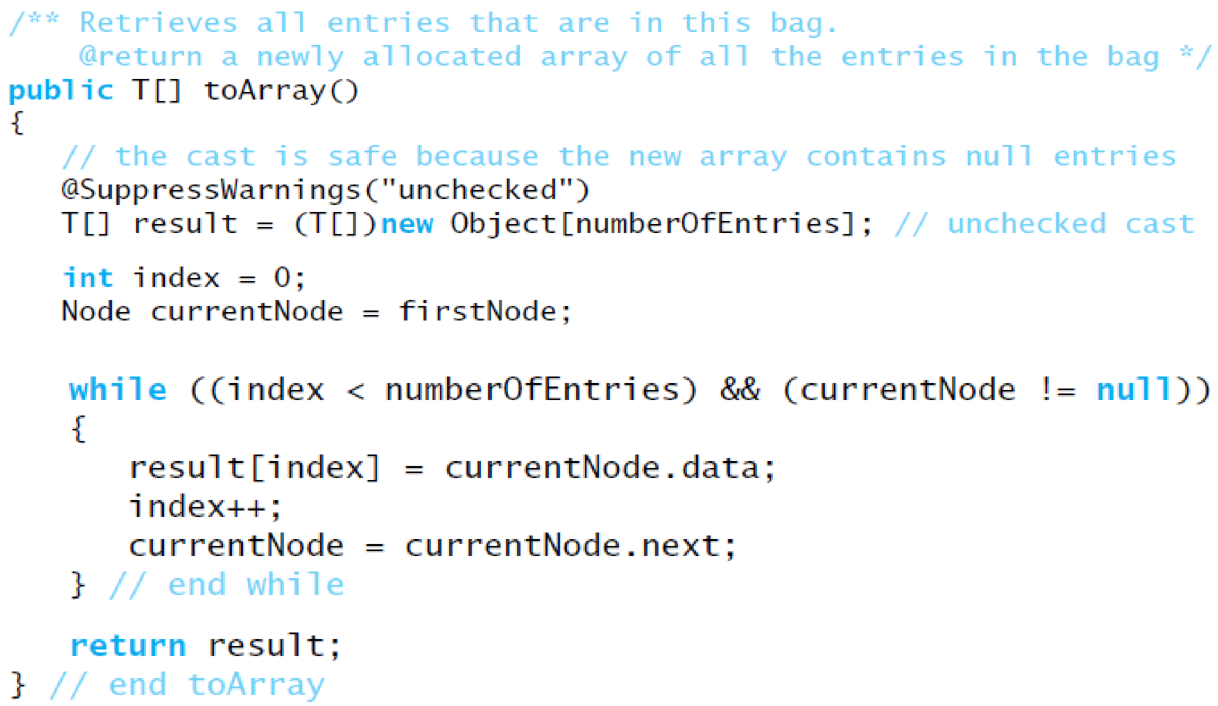
Case I: Add newNode to an empty bag



Case II: Add newNode to non-empty bag



- isEmpty() : check if bag is empty, same as previous method in array
- isFull(): always return false, unless out-of-memory error
- toArray(): // Note: how to traverse a linked list



```
/** Retrieves all entries that are in this bag.
    @return a newly allocated array of all the entries in the bag */
public T[] toArray()
{
   // the cast is safe because the new array contains null entries
   @SuppressWarnings("unchecked")
   T[] result = (T[])new Object[numberOfEntries]; // unchecked cast

   int index = 0;
   Node currentNode = firstNode;

   while ((index < numberOfEntries) && (currentNode != null))
   {
      result[index] = currentNode.data;
      index++;
      currentNode = currentNode.next;
   } // end while

   return result;
} // end toArray
```

- Testing the Incomplete Implementation

  You may write test programs to test completed methods. Other methods specified as stubs to satisfy syntax checker, for example:

```
public T remove()
{
        return null;
}
```

- Sample test program:

```
public static void main (String [] args)
{
   BagInterface < String > myBag = new LinkedBag ();
   System.out.println ("Bag should be empty; isEmpty returns " +
        myBag.isEmpty ());
   System.out.println ("\nTesting add:");
   System.out.println ("Add 15: returns " + myBag.add ("15"));
   System.out.println ("\nBag should not be empty; isEmpty() returns " +
        myBag.isEmpty () + "\n");
   System.out.println ("Add 25: returns " + myBag.add ("25"));
   System.out.println ("Add 35: returns " + myBag.add ("35"));
   System.out.println ("Add 45: returns " + myBag.add ("45"));
   System.out.println ("\nBag should not be empty; isEmpty() returns " +
        myBag.isEmpty ());
   System.out.println ("\nBag should contain\n45 35 25 15 ");
   System.out.println ("\nTesting display():");
   display (myBag);
} // end main
```

- display(BagInterface < String > myBag): use toArray() method

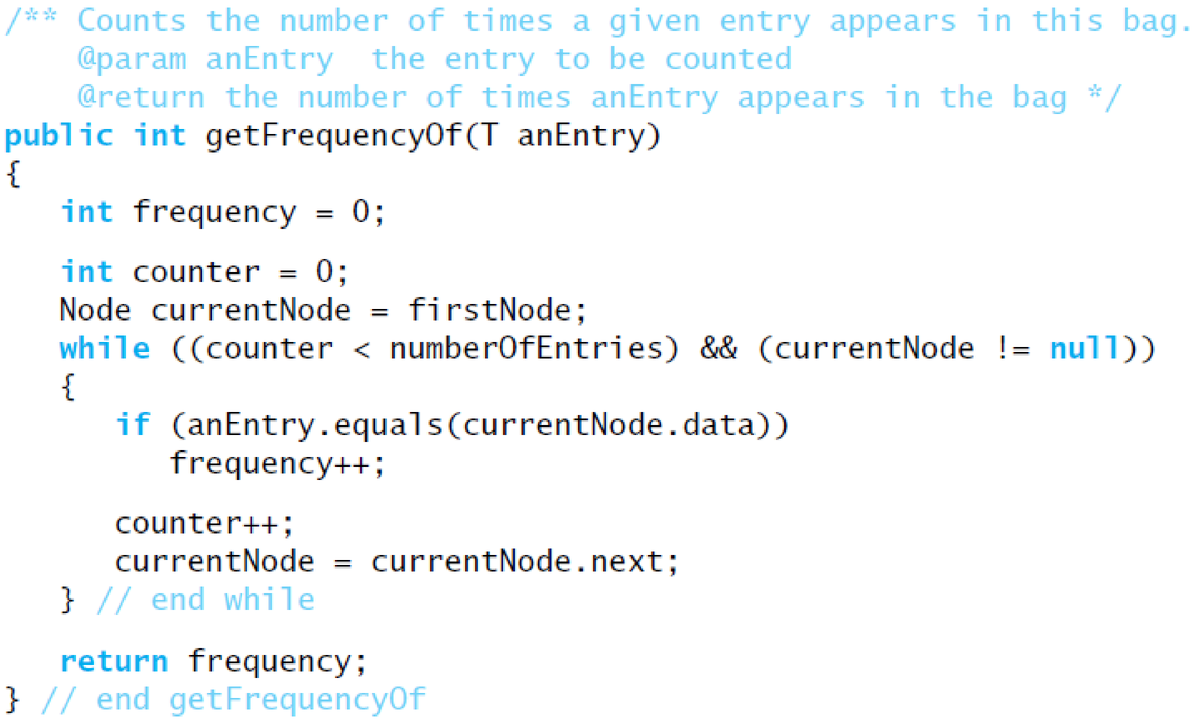
```
        Object [] bagArray = myBag.toArray ();
        for (int index = 0 ; index < bagArray.length ; index++)
        {
           System.out.print (bagArray [index] + " ");
        } // end for
        System.out.println ();
```

- Sample output:

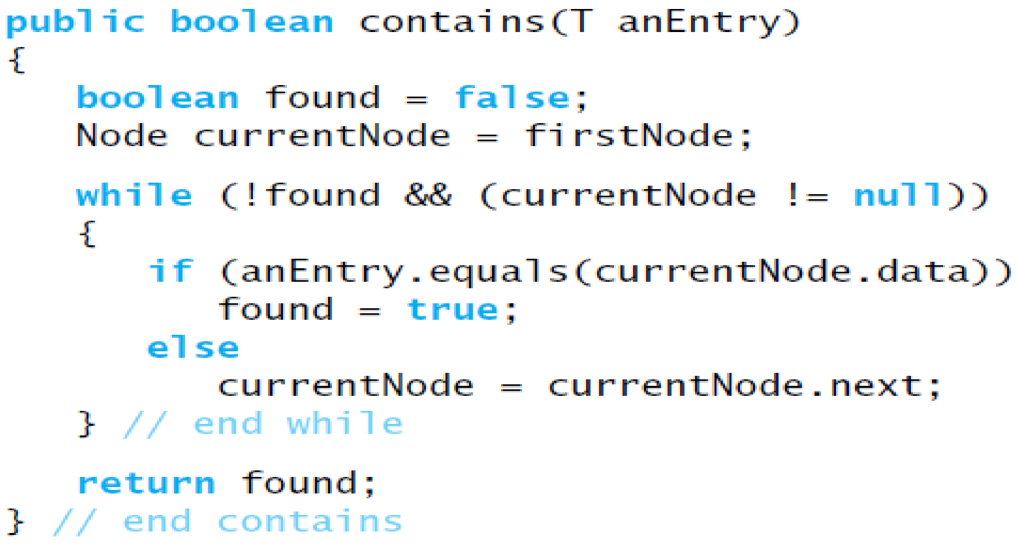
```
Bag should be empty; isEmpty returns true
Testing add:
Add 15: returns true
Bag should not be empty; isEmpty() returns false
Add 25: returns true
Add 35: returns true
Add 45: returns true
Bag should not be empty; isEmpty() returns false
Bag should contain
45 35 25 15
Testing display():
45 35 25 15
```

- Let consider additional methods

// several methods which need to traverse linked list
// Note: you may use “counter” or “null” to check for end of list



```java
/** Counts the number of times a given entry appears in this bag.
    @param anEntry  the entry to be counted
    @return the number of times anEntry appears in the bag */
public int getFrequencyOf(T anEntry)
{
   int frequency = 0;

   int counter = 0;
   Node currentNode = firstNode;
   while ((counter < numberOfEntries) && (currentNode != null))
   {
      if (anEntry.equals(currentNode.data))
         frequency++;

      counter++;
      currentNode = currentNode.next;
   } // end while

   return frequency;
} // end getFrequencyOf
```



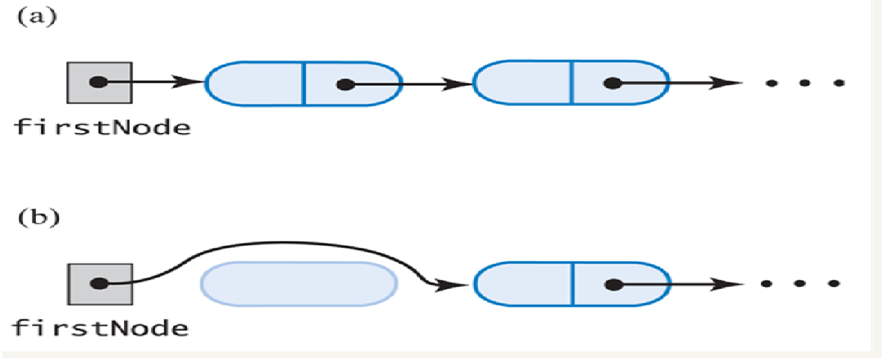
```java
public boolean contains(T anEntry)
{
   boolean found = false;
   Node currentNode = firstNode;

   while (!found && (currentNode != null))
   {
      if (anEntry.equals(currentNode.data))
         found = true;
      else
         currentNode = currentNode.next;
   } // end while

   return found;
} // end contains
```

- Remove method: to remove a node from linked list

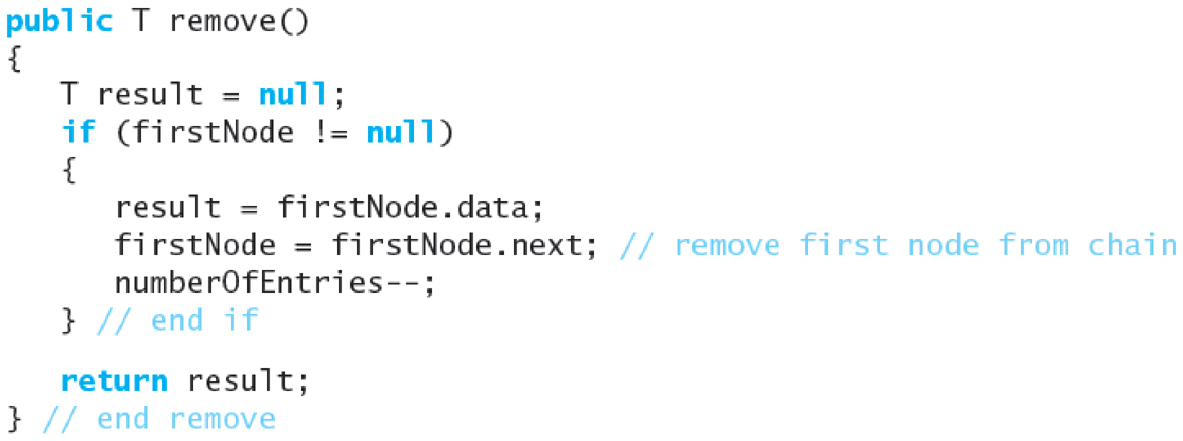
  Note: Java runtime environment automatically deallocates and recycles removed nodes

Remove: easy to remove first node

First = First.Next // set First to second node.

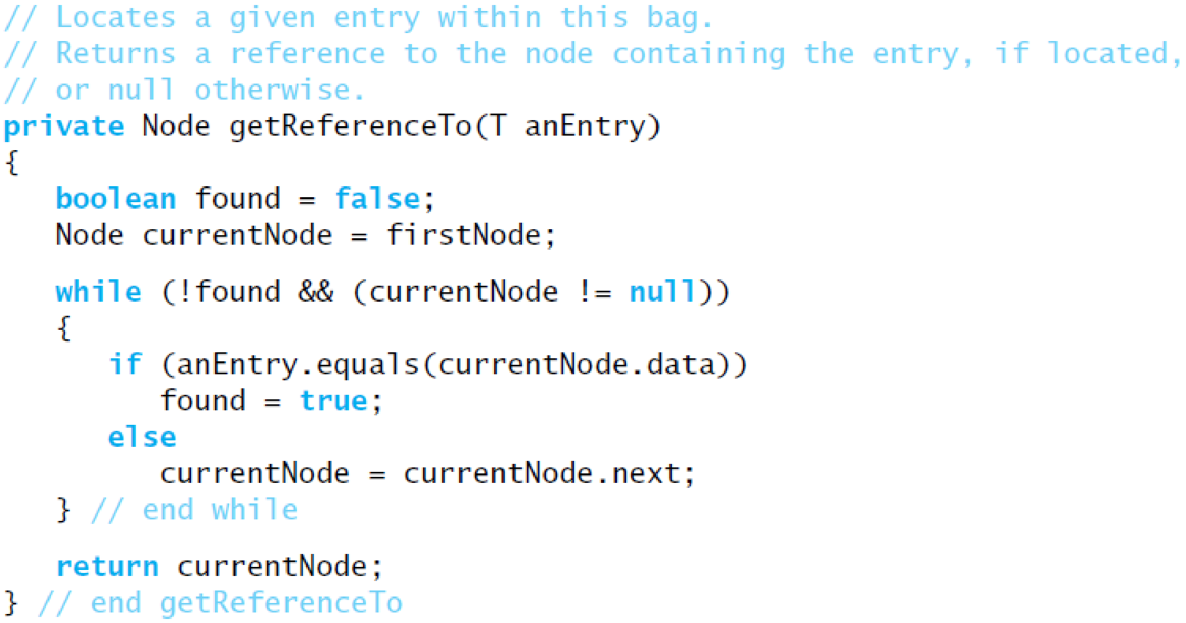


// remove() first node, return date in the node

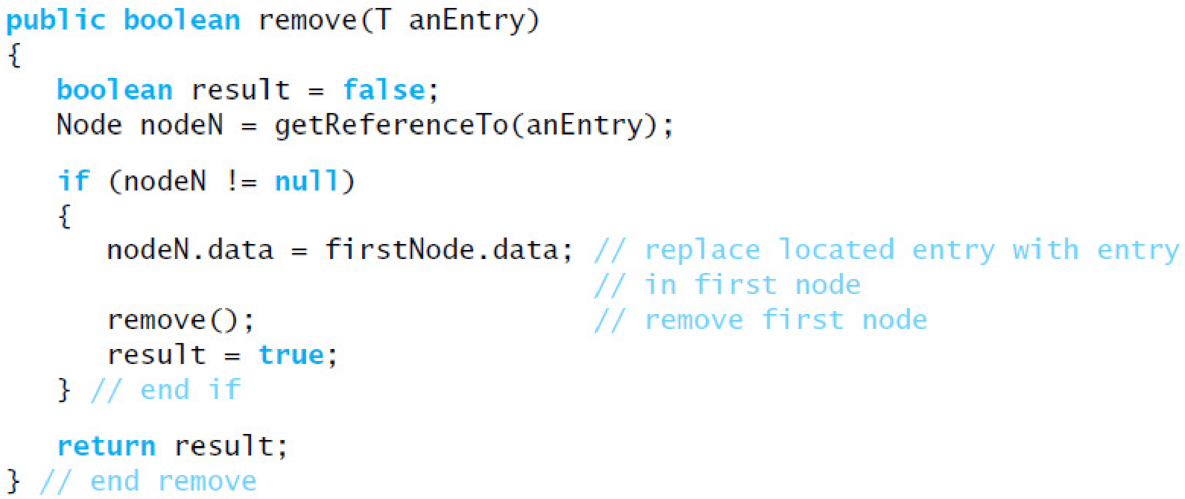


```
public T remove()
{
   T result = null;
   if (firstNode != null)
   {
      result = firstNode.data;
      firstNode = firstNode.next; // remove first node from chain
      numberOfEntries--;
   } // end if

   return result;
} // end remove
```

// remove a given entry
// locate the reference of the desired node, private method
// move first data to desired node, remove 1st node

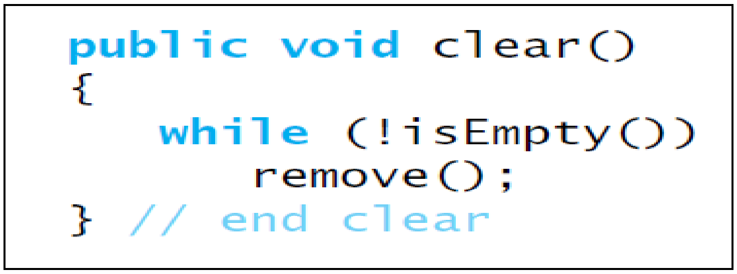


```
// Locates a given entry within this bag.
// Returns a reference to the node containing the entry, if located,
// or null otherwise.
private Node getReferenceTo(T anEntry)
{
   boolean found = false;
   Node currentNode = firstNode;

   while (!found && (currentNode != null))
   {
      if (anEntry.equals(currentNode.data))
         found = true;
      else
         currentNode = currentNode.next;
   } // end while

   return currentNode;
} // end getReferenceTo
```



```
public boolean remove(T anEntry)
{
   boolean result = false;
   Node nodeN = getReferenceTo(anEntry);

   if (nodeN != null)
   {
      nodeN.data = firstNode.data; // replace located entry with entry
                                   // in first node
      remove();                    // remove first node
      result = true;
   } // end if

   return result;
} // end remove
```

// clear() method



```
public void clear()
{
   while (!isEmpty())
      remove();
} // end clear
```

- Update class Node to include set and get methods

  Version 1: As before, as an inner class, i.e. define within Bag class
  The class LBag can access private data fields directly

```
private class Node
{
  private T    data; // entry in Bag
  private Node next; // link to next node

  private Node(T dataPortion)
  {
    data = dataPortion;
    next = null;
  } // end constructor

  private Node(T dataPortion, Node nextNode)
  {
    data = dataPortion;
    next = nextNode;
  } // end constructor

  private T getData()
  {
    return data;
  } // end getData

  private void setData(T newData)
  {
    data = newData;
  } // end setData

  private Node getNextNode()
  {
    return next;
  } // end getNextNode
```

```
 private void setNextNode(Node nextNode)
 {
   next = nextNode;
 } // end setNextNode
} // end Node
```

So, you may update previous program statements to use these new methods.

Examples:

```
result = firstNode.data →result = firstNode.getData();

currentNode = currentNode.next →
          currentNode = currentNode.getNextNode();
```

- Version 2: Define as a class within a package (but not in Bag class)

    - Omit all access modifiers except on data fields
    - Add <T> after each occurrence of Node within the class (except in constructor names)

```
package BagPackage;
class Node<T>
{
  private T       data;
  private Node<T> next;

  Node(T dataPortion) // the constructor name is Node, not Node<T>
  {
    data = dataPortion;
    next = null;
  } // end constructor

  Node(T dataPortion, Node<T> nextNode)
  {
```

```
  data = dataPortion;
  next = nextNode;
 } // end constructor

 T getData()
 {
  return data;
 } // end getData

 void setData(T newData)
 {
  data = newData;
 } // end setData

 Node<T> getNextNode()
 {
  return next;
 } // end getNextNode

 void setNextNode(Node<T> nextNode)
 {
  next = nextNode;
 } // end setNextNode
} // end Node
```

- Require to change LinkedBag to access the class Node. Example:

```
package BagPackage;
public class LinkedBag<T> implements BagInterface<T>
{
 private Node<T> firstNode;   // need <T>  in all statements

   //. . .
   public boolean add (T newEntry)
   {
      Node < T > newNode = new Node < T > (newEntry);
      newNode.setNextNode (firstNode);
      firstNode = newNode;
      numberOfEntries++;
      return true;
   } // end add
   //. . .

} // end LinkedBag
```

- Pros and Cons of Using a Chain
    - Bag can grow and shrink in size as necessary
    - Possible to remove and recycle nodes
    - Copying values for array enlargement not needed
    - Removing specific value entry requires search of array or chain
    - Chain requires more memory than array of same length