



University of
Pittsburgh

Algorithms and Data Structures 1

CS 0445



Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines:
 - Homework 2: this Friday @ 11:59 pm
 - Lab 1: next Monday @ 11:59 pm
 - Programming Assignment 1: Friday Oct. 7th
- Draft slides and handouts available on Canvas
- Lecture recordings are available under Panopto Video on Canvas
- **Student Support Hours** of the teaching team are posted on the Syllabus page

Previous Lecture ...

- ADT Bag Implementations
 - Fixed-size array: ArrayBag
 - `getFrequencyOf(T)`, `contains(T)`, `remove()`, `remove(T)`
 - Resizable array: ResizableArrayBag

Muddiest Points

- **Q: The book shows that the return type of "remove anEntry" is a boolean. Why? The code we worked on in class returned the Entry**
- **A: There are different ways of *designing* a method. The book uses a Boolean to return false when the entry is not found; we return the removed entry (not possible in the book's design) with null returned when the entry is not found.**
- **Q: I don't understand why we need to do unchecked casts for arrays of type T when type erasure happens. Why would we not declare an array of type Object instead?**
- **A: We want to make the compiler happy when it does type checking. Type checking happens **before** type erasure.**

Muddiest Points

- **Q: I'm still confused about <T> vs <?> vs <Object>, what the hierarchy is and when you would want to use each one.**
- **A:**
 - ArrayBag<T> is a subtype of ArrayBag<?>.
 - ArrayBag<Object> is a subtype of ArrayBag<?>
 - ArrayBag<T> is NOT a subtype of ArrayBag<Object>
 - All of the above are subtypes of Object.
 - Let's say that we are designing a method **void display** that takes an ArrayBag as a parameter.
 - **void display(ArrayBag<T>)** → we need to use T in the method declaration and/or method body
 - **void display(ArrayBag<?>)** → we don't need to use T
 - **void display(ArrayBag<Object>)** → This is limited to only receive ArrayBag<Object>, not for example ArrayBag<Integer>

Muddiest Points

- **Q: Why does the order of an arraybag not matter**
- **A: This is the definition of the ADT Bag. The client (user of a Bag) doesn't expect a Bag to keep its items in a particular order**

Muddiest Points

- **Q: The hardest thing for me to understand is what happens to everything/where all the stuff goes during the remove method.**
- A: The steps we took in remove
 - find the index of an item that we want to remove
 - if not found, return null
 - if found,
 - save the found item by **result = bag[index]**
 - replace the item by the last item of the array
 - **bag[index] = bag[numberOfItems-1]**
 - remove the last item in the array **bag[numberOfItems-1] = null**
 - decrement the logical size of the array **numberOfItems--**

Muddiest Points

- **Q: Why do we care if the one problem method was public? Does revealing we are using an array even matter?**
- **A: Yes, it does!** A Bag maintains its items in no particular order. Allowing the client to get the index of an item implicitly promises the client that the item will remain at that index; which is not guaranteed.

Today's Agenda

- ADT Bag Implementations
 - Fixed-size array: ArrayBag
 - copy constructor
 - Resizable array: ResizableArrayBag
 - add
 - Linked implementation

Copy constructor for ArrayBag<T>

- deep copy! (not deeper though; why?)
- how would you make it a shallow copy?

```
//Copy constructor
public ArrayBag(ArrayBag<T> other){
    checkCapacity(other.bag.length);
    @SuppressWarnings("unchecked")
    T[] temp = (T[]) new Object[other.bag.length];
    for(int i=0; i<other.size; i++){
        temp[i] = (T)other.bag[i];
    }
    bag = temp;
    size = other.size;
    initialized = true;
}
```

ResizableArrayBag: add

- Should we double the capacity before or after adding the item to the array?

```
public boolean add(T item) {
    checkIntegrity();
    boolean result = false;
    if(!isFull()){
        bag[size] = item;
        size++;
        result = true;
    }

    if(size == bag.length){
        doubleCapacity();
    }
    return result;
}
```

ResizableArrayBag: doubleCapacity

```
private void doubleCapacity(){
    int capacity = bag.length;
    checkCapacity(2*capacity);

    @SuppressWarnings("unchecked")
    T[] temp = (T[])new Object[2*capacity];
    for(int i=0; i<size; i++){
        temp[i] = bag[i];
    }
    bag= temp;
    //bag = Arrays.copyOf(bag, 2*capacity);
}
```

ResizableArrayBag

- Can we still have the final keyword for the underlying array?

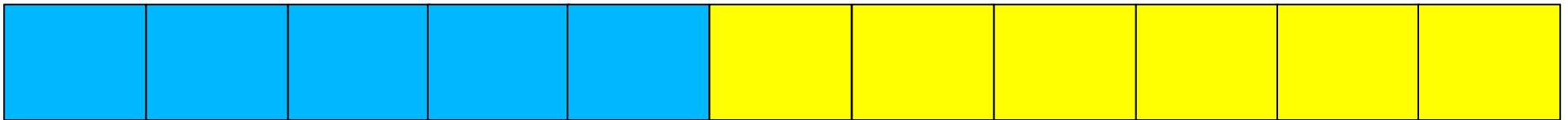
```
private T[] bag;
```

Pros and Cons of Using an Array

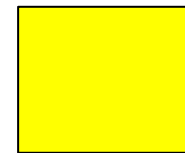
- Pros:
 - Adding an entry to the bag is fast
 - Removing an unspecified entry is fast
- Cons:
 - May be wasteful of memory
 - for example, we have 10 items now in the Bag, but we *expect* that we will have 1,000,000 items later
 - how big should the array be?
 - Increasing the size of the array requires time to copy its entries

Linked Implementation

Array



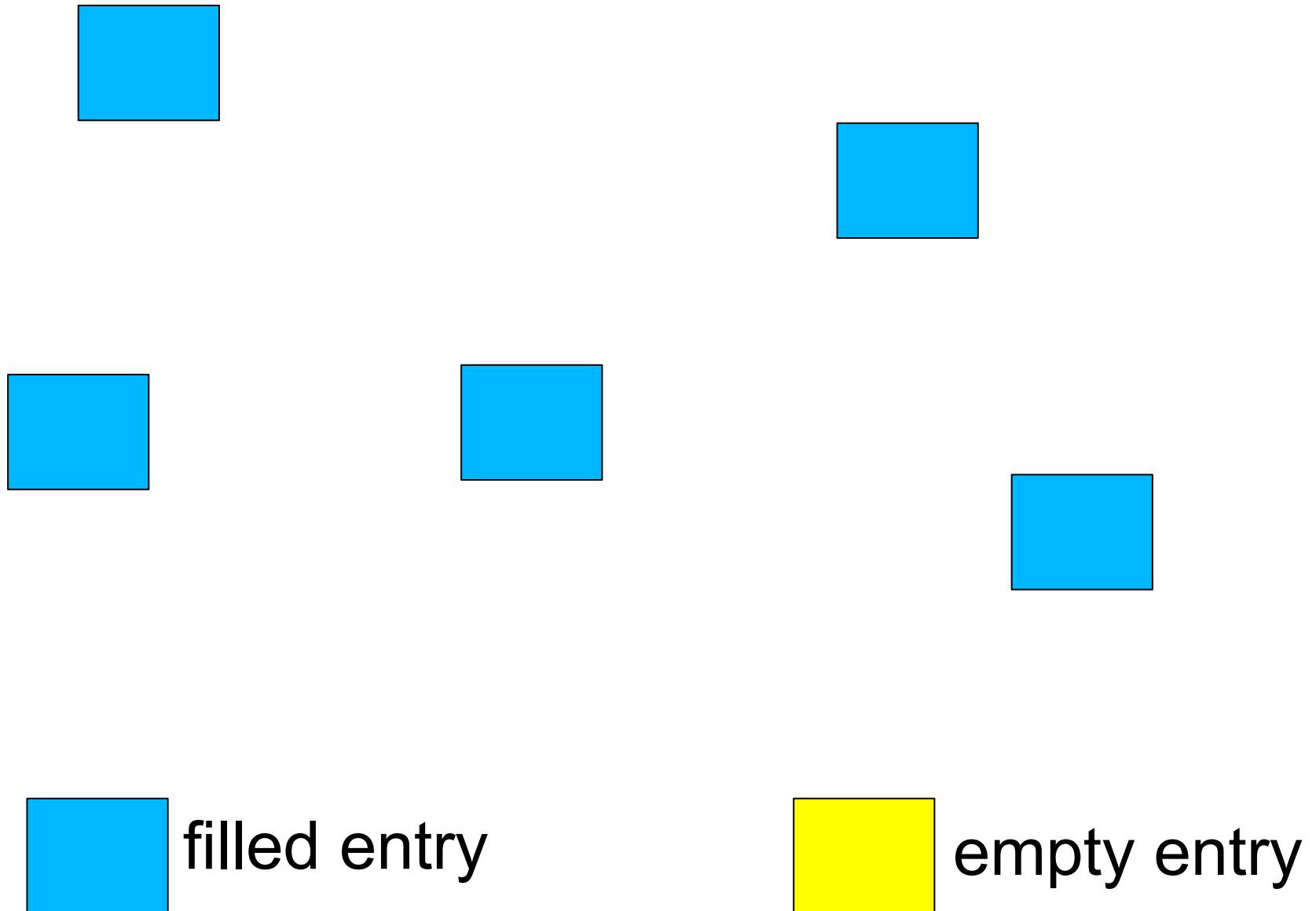
filled entry



empty entry

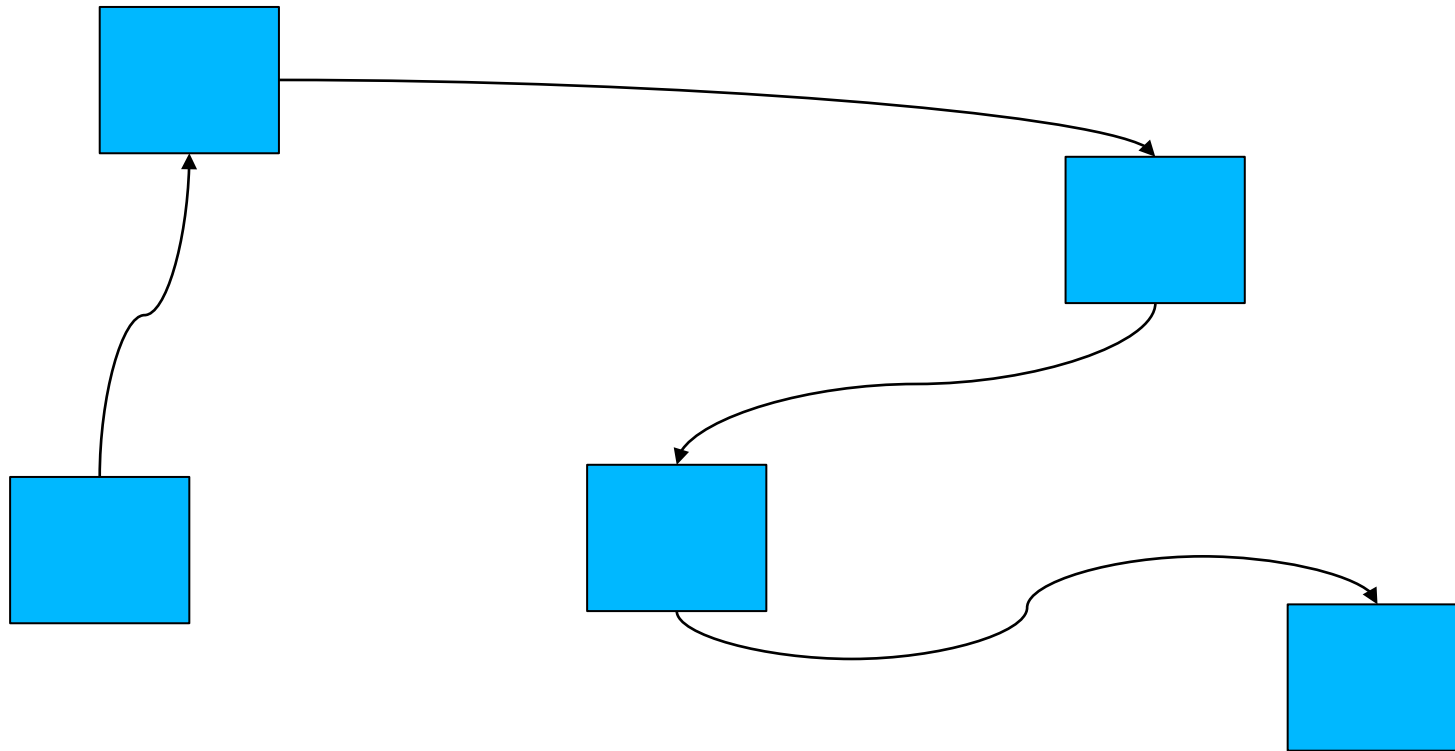
Linked Implementation

Linked Chain



Linked Implementation

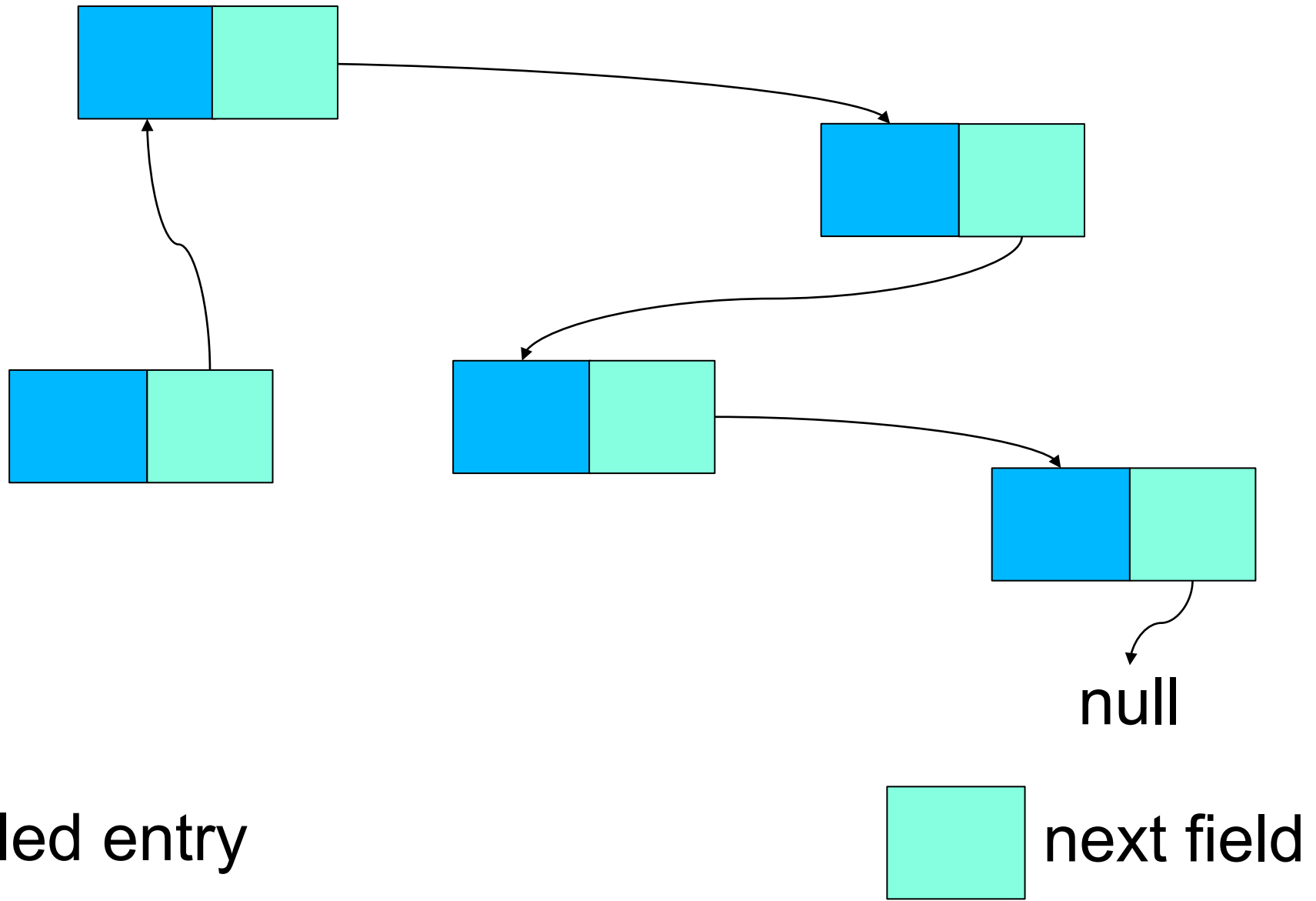
Linked Chain



 filled entry

Linked Implementation

Linked Chain



Pros of Using a Linked Chain

- Bag can grow and shrink in size as necessary.
- Remove and recycle nodes that are no longer needed
 - Using Java's garbage collection

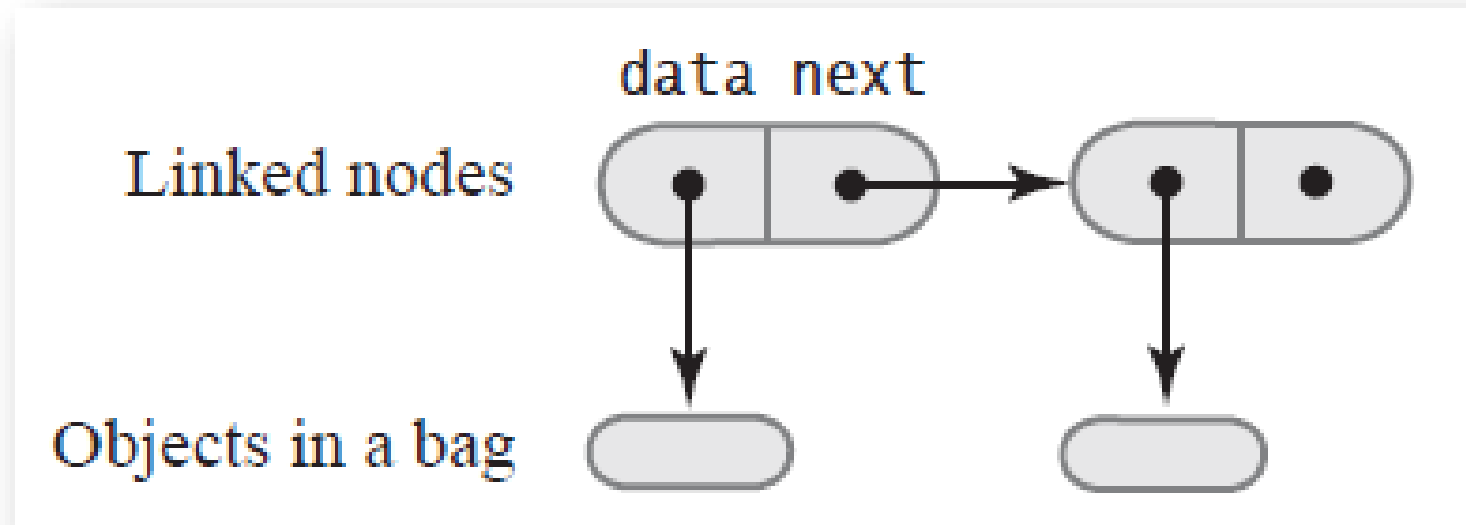
The Private Class Node

- The private inner class **Node**

```
1 private class Node
2 {
3     private T    data; // Entry in bag
4     private Node next; // Link to next node
5
6     private Node(T dataPortion)
7     {
8         this(dataPortion, null);
9     } // end constructor
10
11     private Node(T dataPortion, Node nextNode)
12     {
13         data = dataPortion;
14         next = nextNode;
15     } // end constructor
16 } // end Node
```

The Private Class Node

Two linked nodes that each **references** object data



Class LinkedBag

We need to keep track of only the first node in the chain!

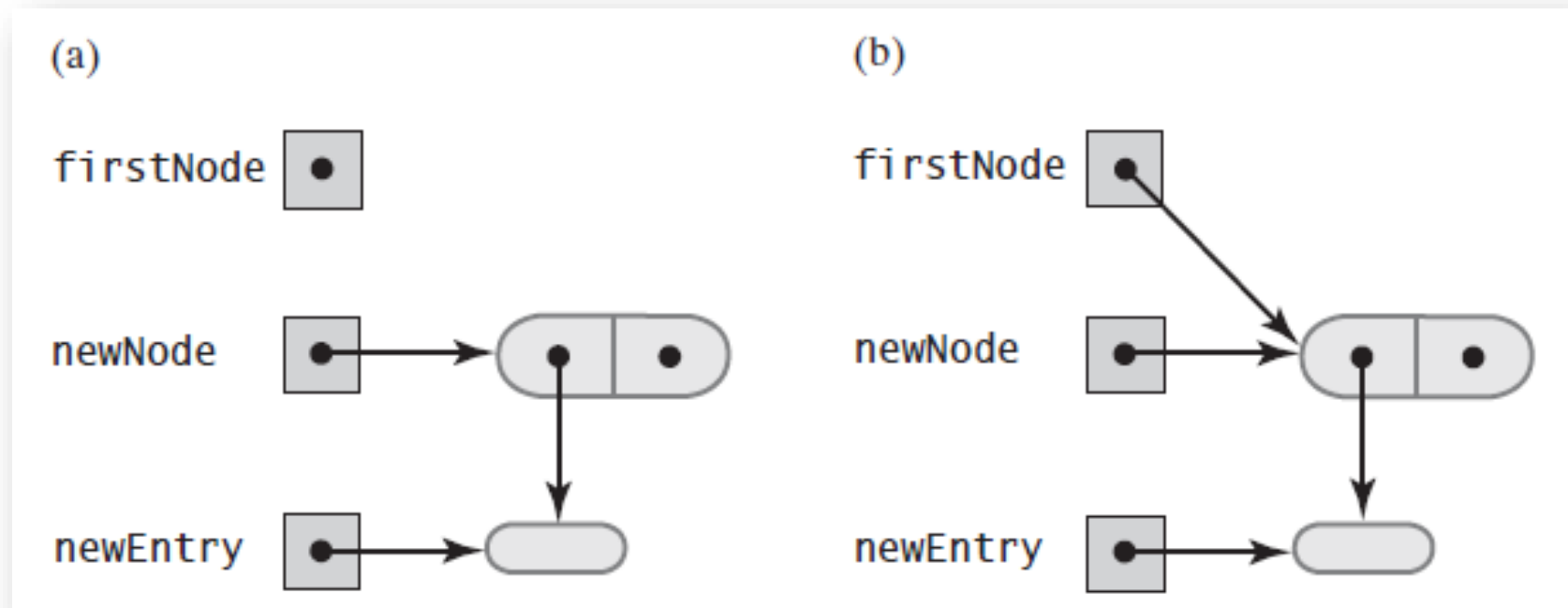
```
1  /**
2   * A class of bags whose entries are stored in a chain of linked nodes.
3   * The bag is never full.
4   * @author Frank M. Carrano
5   */
6  public final class LinkedBag<T> implements BagInterface<T>
7  {
8      private Node firstNode;      // Reference to first node
9      private int  numberOfEntries;
10
11     public LinkedBag()
12     {
13         firstNode = null;
14         numberOfEntries = 0;
15     } // end default constructor
16
17     Implementations of the public methods declared in BagInterface go here.
```

Class `LinkedList`

```
14     numberOfEntries = 0;
15 } // end default constructor
16
17 < Implementations of the public methods declared in BagInterface go here. >
18
19 . . .
20
21 private class Node // Private inner class
22 {
23     < See Listing 3-1. >
24 } // end Node
25 } // end LinkedList
```

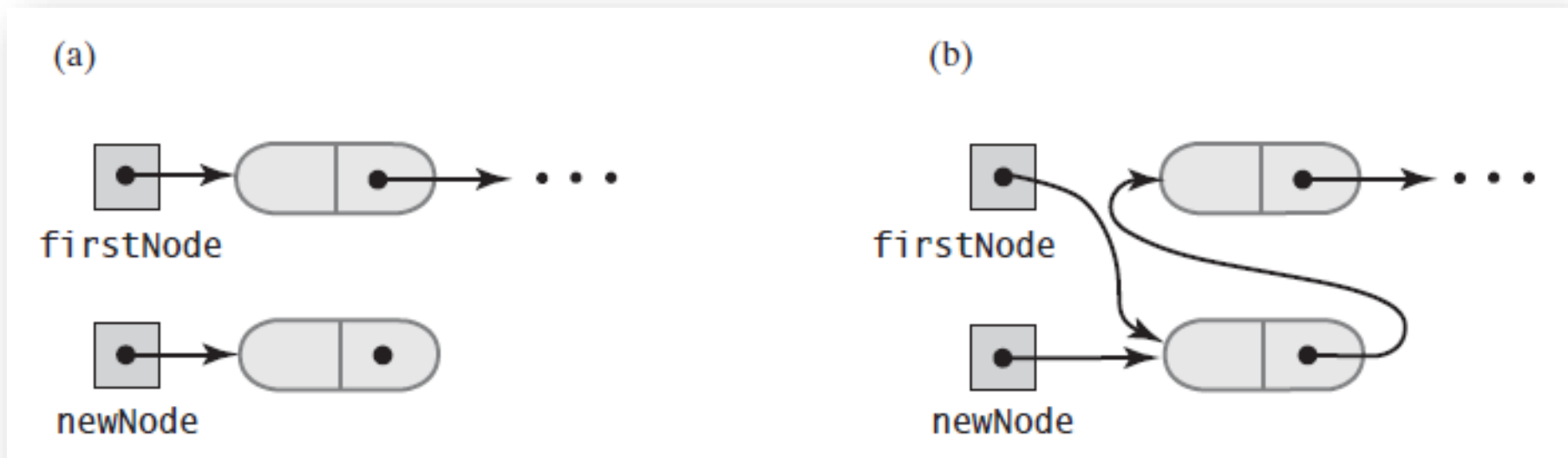
Beginning a Chain of Nodes

(a) An empty chain and a new node; (b) after adding a new node to a chain that was empty



Beginning a Chain of Nodes

A chain of nodes (a) just prior to adding a node at the beginning; (b) just after adding a node at the beginning



LinkedBag.add

- The method **add**

```
/** Adds a new entry to this bag.
 * @param newEntry The object to be added as a new entry.
 * @return True. */
public boolean add(T newEntry) // OutOfMemoryError possible
{
    // Add to beginning of chain:
    Node newNode = new Node(newEntry);
    newNode.next = firstNode;    // Make new node reference rest of chain
                                // (firstNode is null if chain is empty)
    firstNode = newNode;        // New node is at beginning of chain
    numberOfEntries++;
    return true;
} // end add
```

Method toArray

- The method `toArray` returns an array of the entries currently in a bag by **traversing** the chain

```
/** 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
```

Method getFrequencyOf

- Counts the number of times a given entry appears
- Also traverses the chain

```
/** 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 loopCounter = 0;
    Node currentNode = firstNode;
    while ((loopCounter < numberOfEntries) && (currentNode != null))
    {
        if (anEntry.equals(currentNode.data))
            frequency++;
        loopCounter++;
        currentNode = currentNode.next;
    } // end while
    return frequency;
} // end getFrequencyOf
```

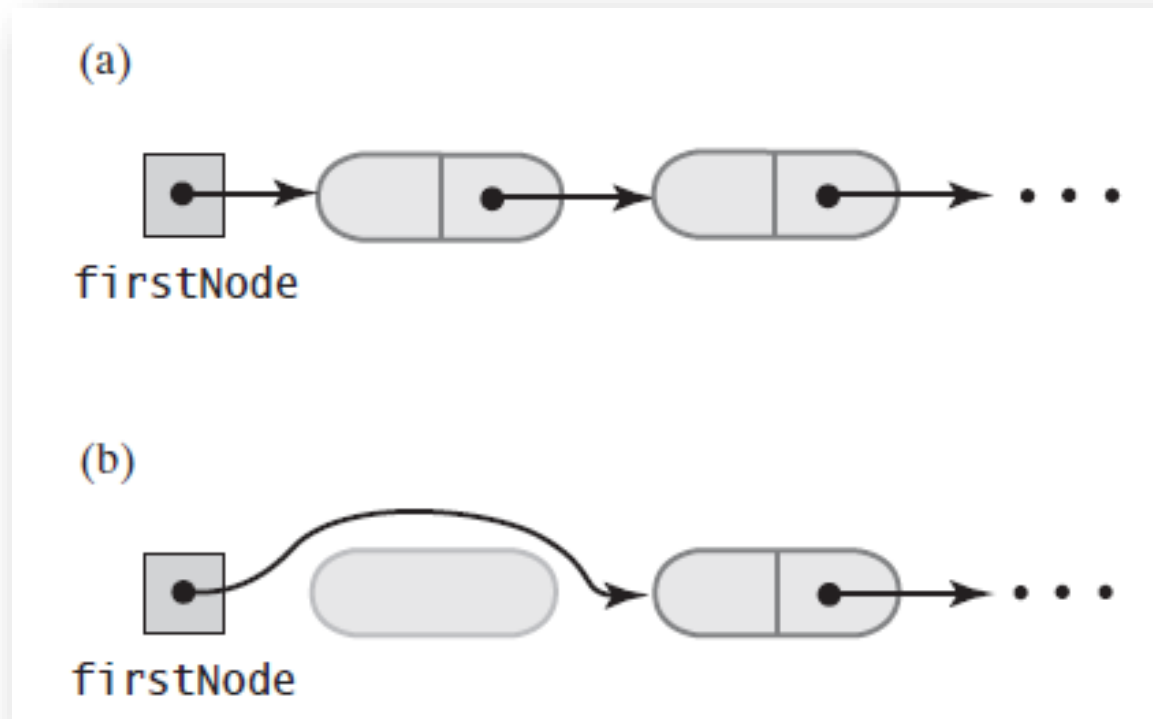
Method contains

- Determine whether a bag contains a given entry

```
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
```

Removing an unspecified item

A chain of nodes (a) just prior to removing the first node; (b) just after removing the first node



Removing a specified item

- Note need for private method `getReferenceTo`
- Returns a reference to a node that references an object that equals anEntry

```
// 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
```

Method `remove`

- Similar trick to what we did in `ArrayBag.remove(T)`
 - replace data by data of first item
- Note use of method `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
        firstNode = firstNode.next; // Remove first node
        numberOfEntries--;
        result = true;
    } // end if
    return result;
} // end remove
```


Node as a Public class

- **Node** can be implemented as an independent class
- Needs to be generic!

```
2 class Node<T>
3 {
4     private T      data;
5     private Node<T> next;
6
7     Node(T dataPortion) // The constructor's name is Node, not Node<T>
8     {
9         this(dataPortion, null);
10    } // end constructor
```

Node as a Public class

- Need setters and getters

```
12     Node(T dataPortion, Node<T> nextNode)
13     {
14         data = dataPortion;
15         next = nextNode;
16     } // end constructor
17
18     T getData()
19     {
20         return data;
21     } // end getData
22
23     void setData(T newData)
24     {
25         data = newData;
26     } // end setData
```

Node as a Public class

```
23 void setData(T newData)
24 {
25     data = newData;
26 } // end setData
27
28 Node<T> getNextNode()
29 {
30     return next;
31 } // end getNextNode
32
33 void setNextNode(Node<T> nextNode)
34 {
35     next = nextNode;
36 } // end setNextNode
37 } // end Node
```

When **Node** is a Public class

```
public class LinkedBag<T> implements BagInterface<T>
{
    private Node<T> firstNode;
    . . .
    public boolean add(T newEntry)
    {
        Node<T> newNode = new Node<T>(newEntry);
        newNode.setNextNode(firstNode);
        firstNode = newNode;
        numberOfEntries++;

        return true;
    } // end add
    . . .
} // end LinkedBag
```

This occurrence of T is optional

