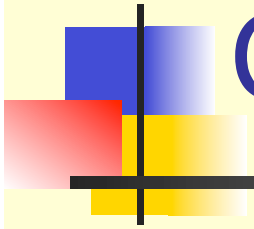
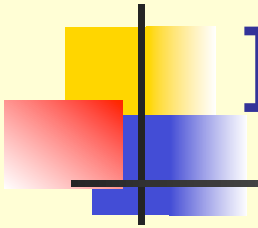


COSC 2006: Data Structures I

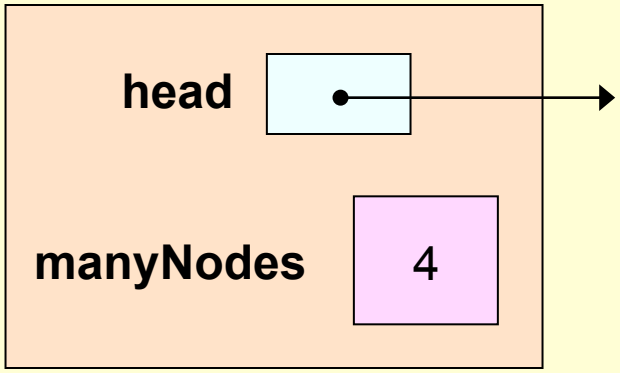
The IntLinkedBag class
uses IntNode for implementation



IntArrayBag and IntLinkedBag

- We have implemented the Bag ADT of integers using an array: IntArrayBag
- We can also implement the Bag ADT of integers using our IntNode class to represent the bag as a linked list

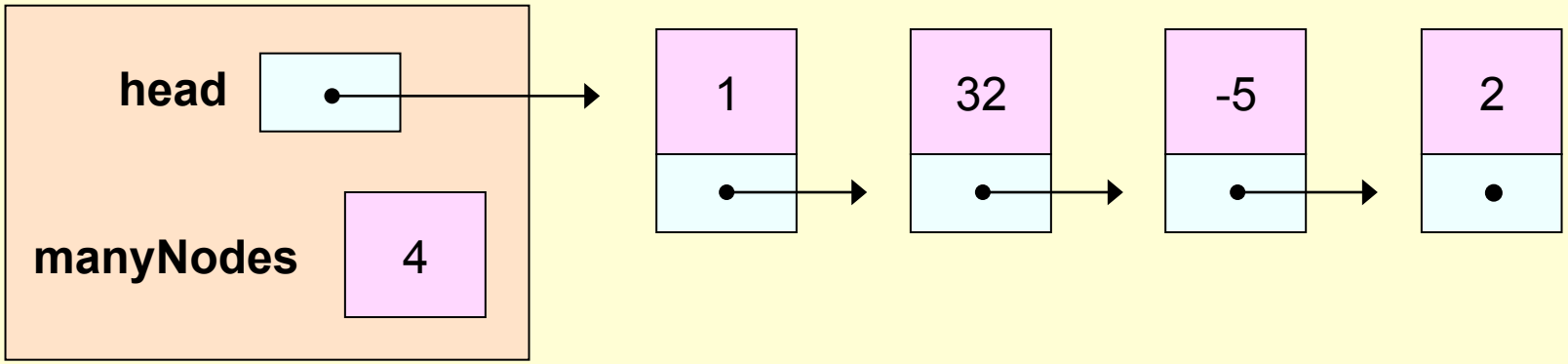
IntLinkedBag objects

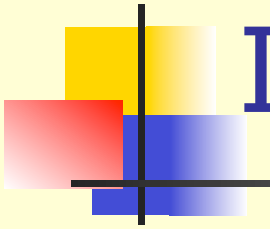


An `IntLinkedBag` object showing the number of nodes in the bag and a reference to the head of the list that contains these nodes

```
private IntNode head;
private int manyNodes;
```

`IntLinkedBag` object showing the list of elements

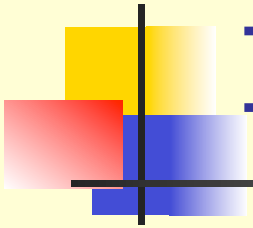




IntLinkedBag class outline

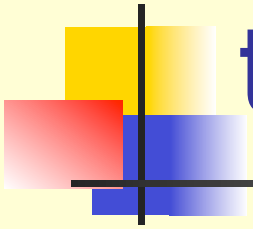
LinkedBag needs only a default constructor. There is no need for an initialCapacity argument or ensureCapacity or trimToSize methods since size is determined by the number of nodes

```
public class IntLinkedBag implements Cloneable
{
   private IntNode head;
   private int manyNodes;

   public IntLinkedBag()
   {
      head = null;
      manyNodes = 0;
   }
   // methods go here
}
```



IntLinkedBag invariant

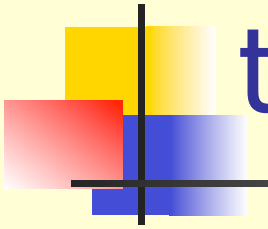
- The elements of the bag are stored in a linked list
- The head reference of the list is stored in the instance variable **`head`**
- The total number of elements in the list is stored in the instance variable **`manyNodes`**



the add and size methods

Add an element to an IntLinkedBag. Here we add at the head of the list.

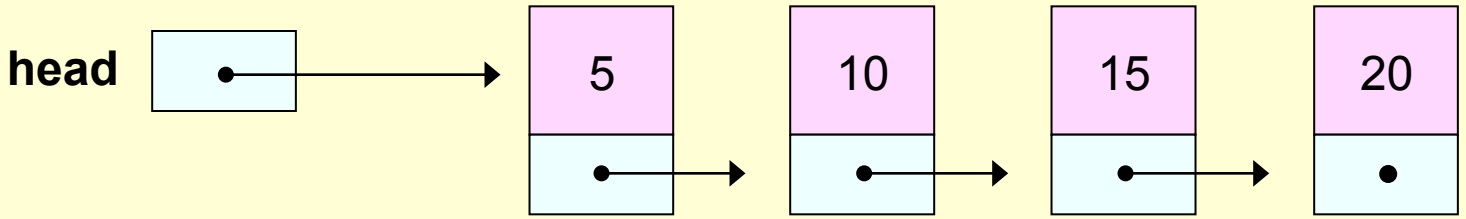
```
public void add(int element)
{
   head = new IntNode(element, head);
   manyNodes++;
}

public int size()
{
   return manyNodes;
}
```



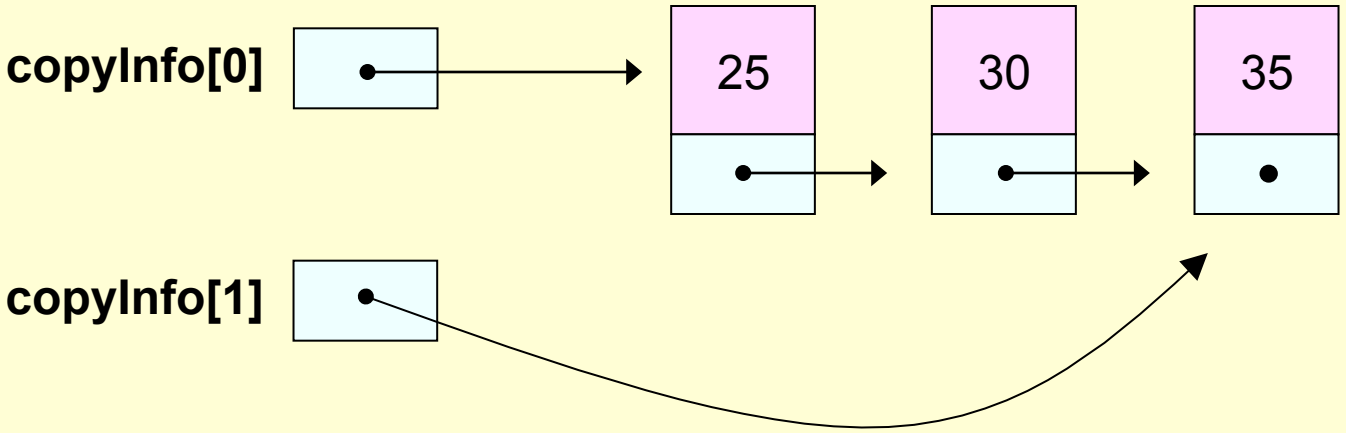
the addAll method (1)

- Method prototype is
  - `public void addAll(IntLinkedBag addend);`
- We can implement this by first making a copy of **`addend`** using **`listCopyWithTail`**
- Then we can link the copy with head of "this bag"
- Finally we can link head of "this bag" to the head of the addend copy

the addAll method (2)

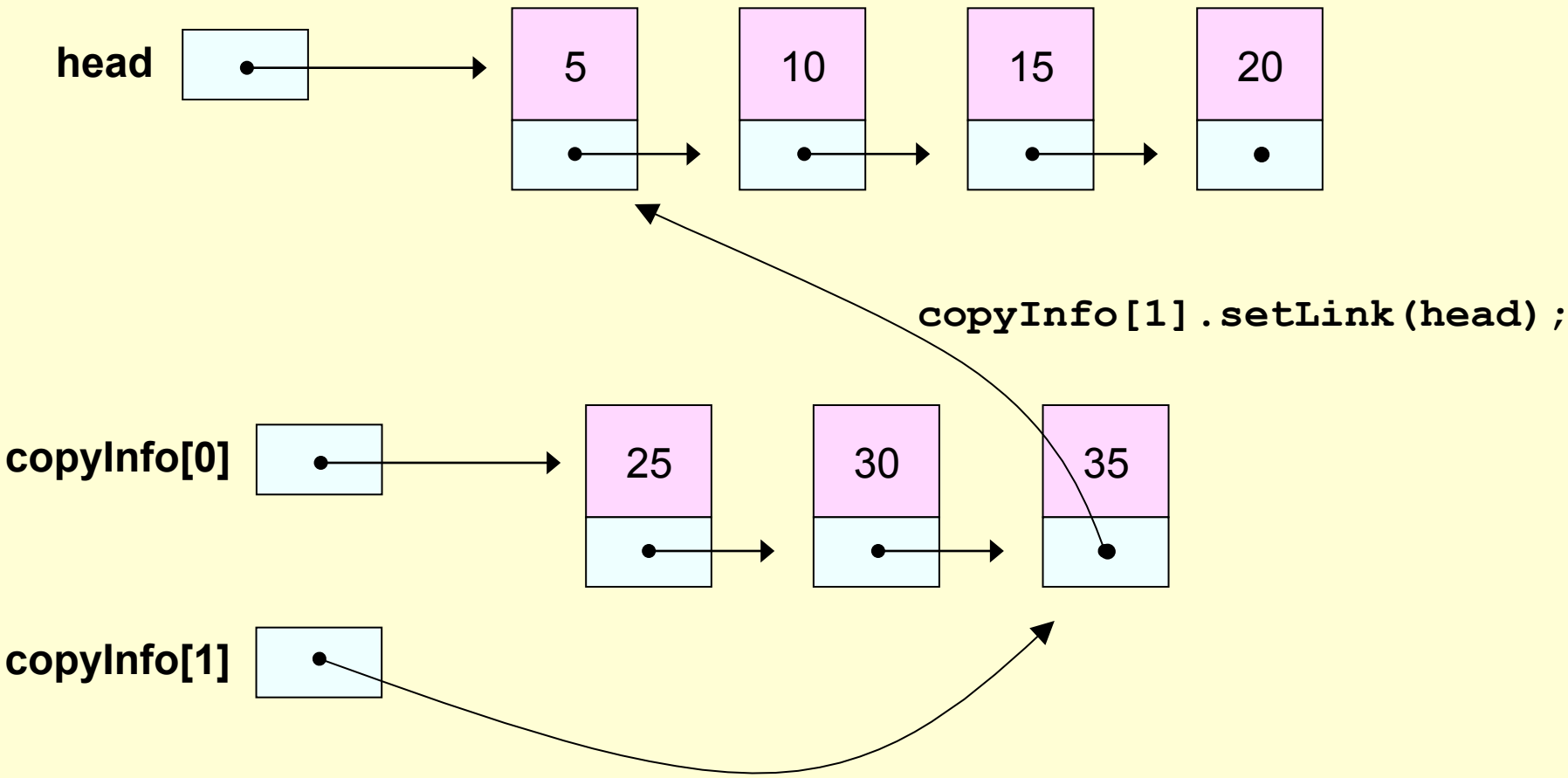



```
IntNode[] copyInfo = IntNode.listCopyWithTail(addend.head);
```

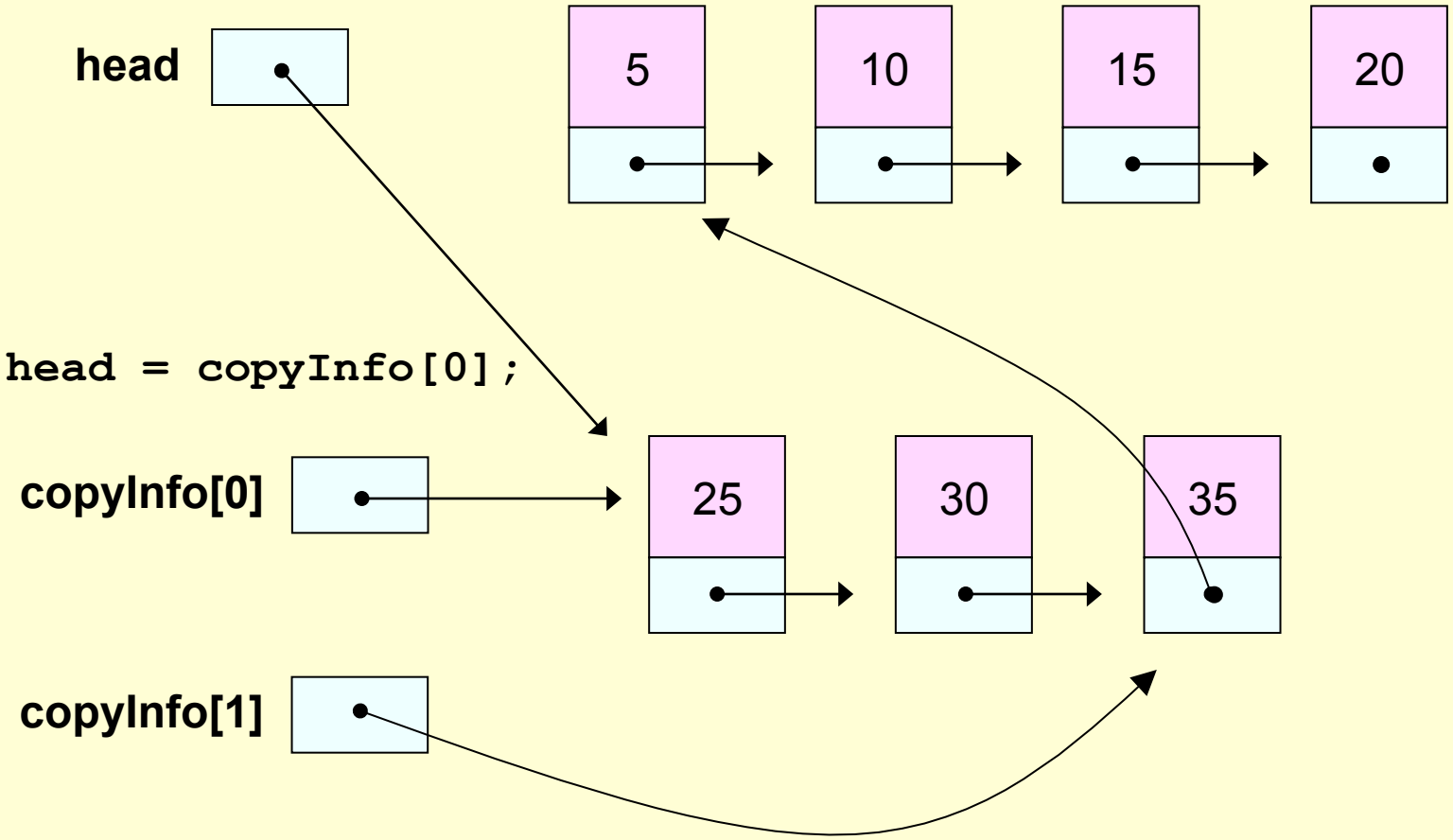


This is the copy of addend bag made by listCopyWithTail

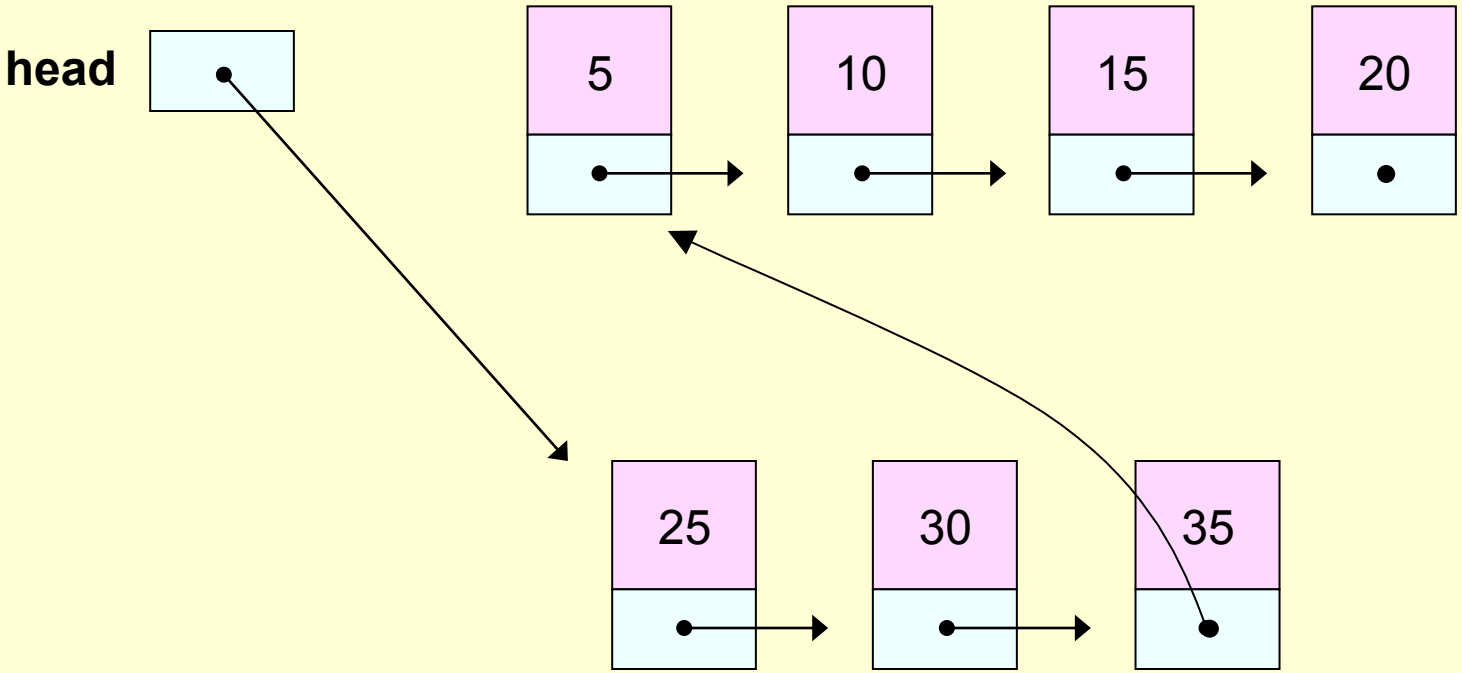
the addAll method (3)



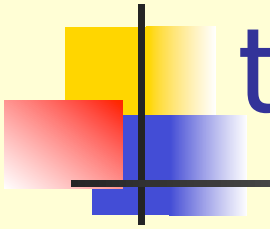
the addAll method (4)



the addAll method (5)



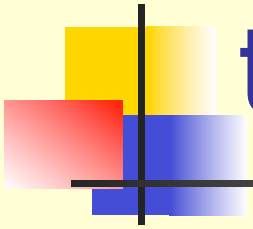
Final result is a list of 7 elements



the addAll method (6)

Add the contents of another bag to this bag.
We can do this using listCopyWithTail

```
public void addAll(IntLinkedBag addend)
{
   if (addend == null)
      throw new IllegalArgumentException("...");
   if (addend.manyNodes > 0)
   {
      IntNode[] copyInfo
         IntNode.listCopyWithTail(addend.head);
      copyInfo[1].setLink(head);
      head = copyInfo[0]; // set "this" head
      manyNodes += addend.manyNodes;
   }
}
```



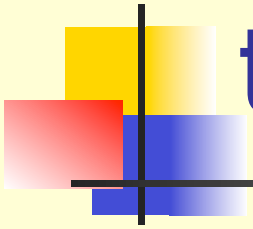
the addMany method

Add new elements to this bag.
We can do this add and the new Java 5 for loop

```
public void addMany(int ... elements)
{
   for (int i: elements)
      add(i);
}
```

The old way of doing this for loop is

for (int k = 0; k < elements.length; k++)
* add(element[k]);*

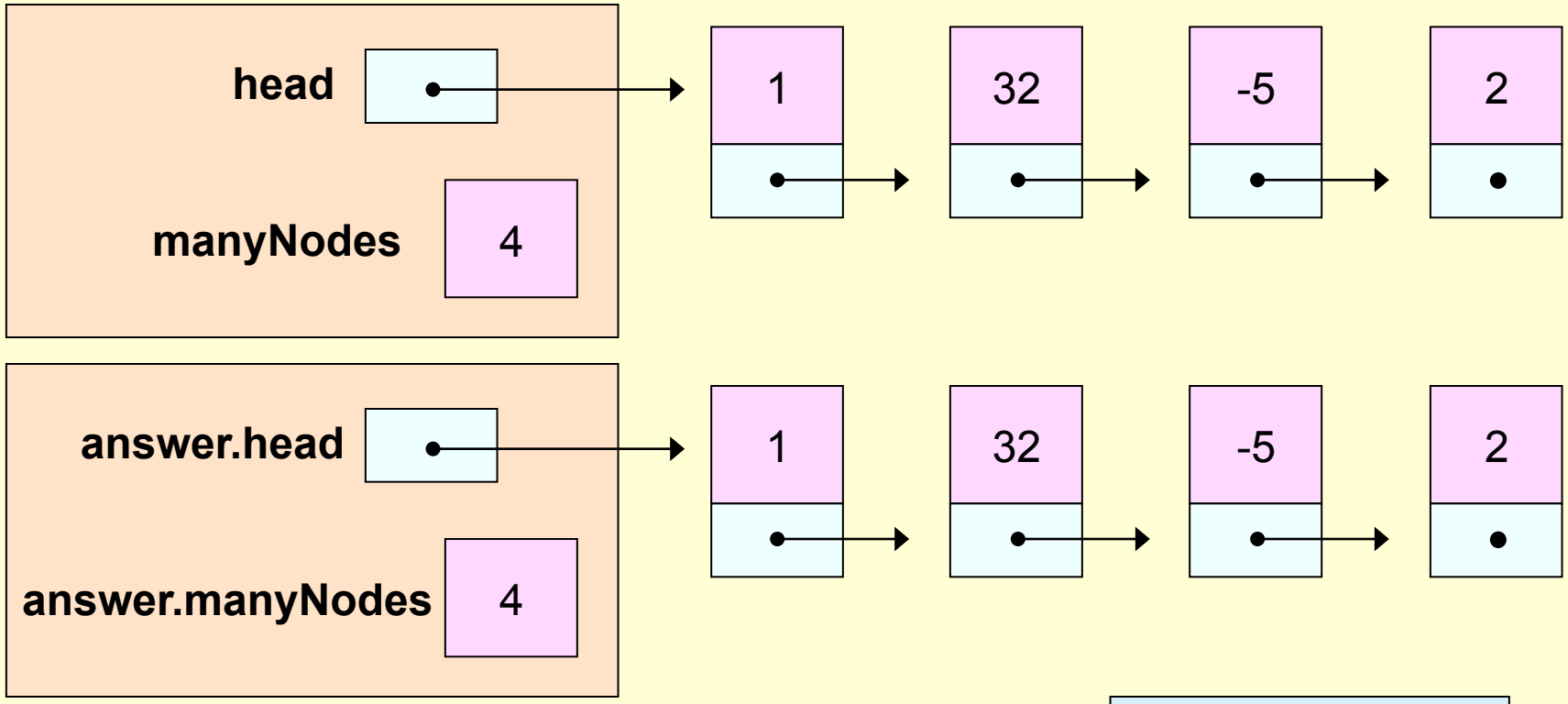


the union method

Add the contants of two bags and return the resulting bag.
The input bags are unchanged and a new bag is created.

```
public static IntLinkedBag union(
   IntLinkedBag b1, IntLinkedBag b2)
{
   if (b1 == null)
      throw new IllegalArgumentException("...");
   if (b2 == null)
      throw new IllegalArgumentException("...");
   IntLinkedBag answer = new IntLinkedBag();
   answer.addAll(b1);
   answer.addAll(b2);
   return answer;
}
```

Cloning an IntLinkedBag

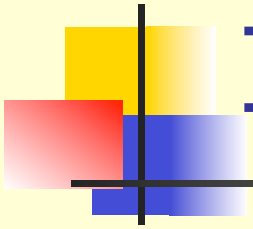



```
answer = (IntLinkedBag) super.clone();
```

clone instance data fields

```
answer.head = IntNode.listCopy(head);
```

clone the list



IntLinkedList clone method

Here we do a deep clone (primitive types)
(1) clone the instance data fields
(2) clone the list

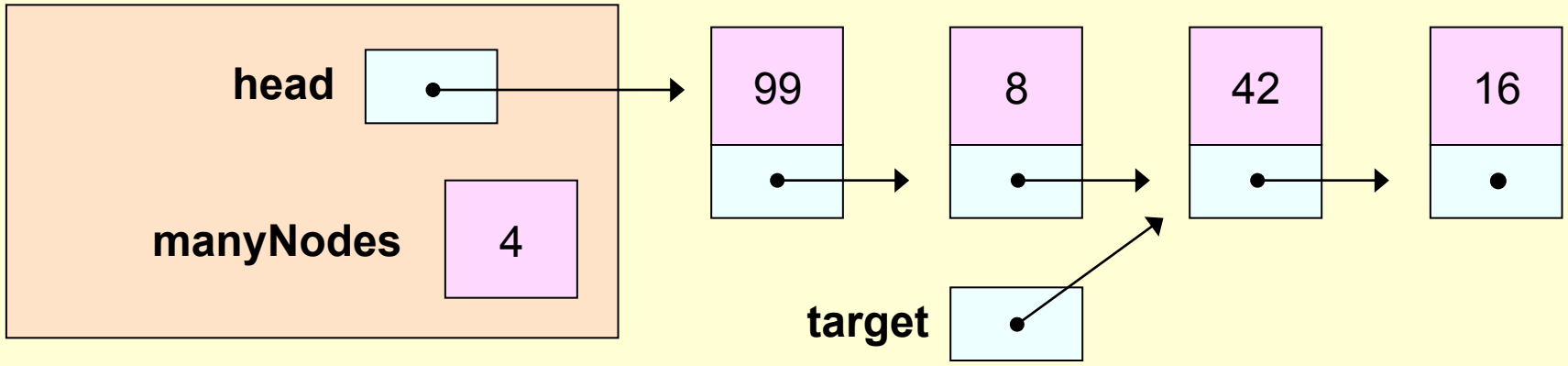
```
public IntLinkedBag clone()
{   IntLinkedBag answer;
    try
    {   answer = (IntLinkedBag) super.clone();
    }
    catch (CloneNotSupportedException e)
    {   throw new RunTimeException("...");
    }
    answer.head = IntNode.listCopy(head);
    return answer;
}
```

The remove method (1)

If we want to remove node with data 42 we can first use listSearch to find this node (target is a reference to it);

```
IntNode target = IntNode.listSearch(head, target);
```

We run into a problem: To remove the node referenced by target we would need a reference to the previous node so we could link the node containing 8 to the node containing 16. But listSearch doesn't give us the previous reference.

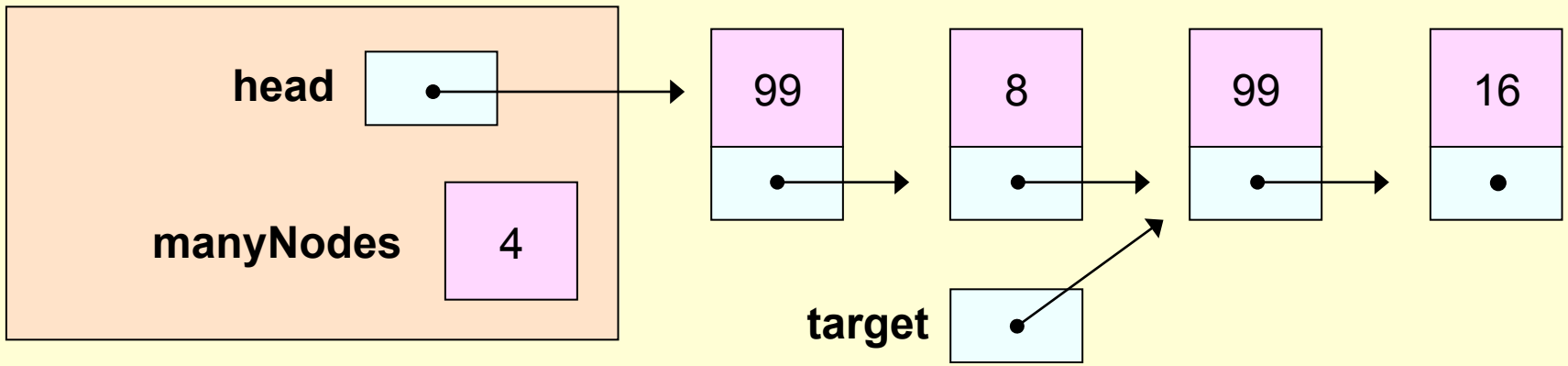


The remove method (2)

Since there is no ordering to the elements in a bag we can solve this problem with the following trick:

First copy the data (99) from the head node to the target node using

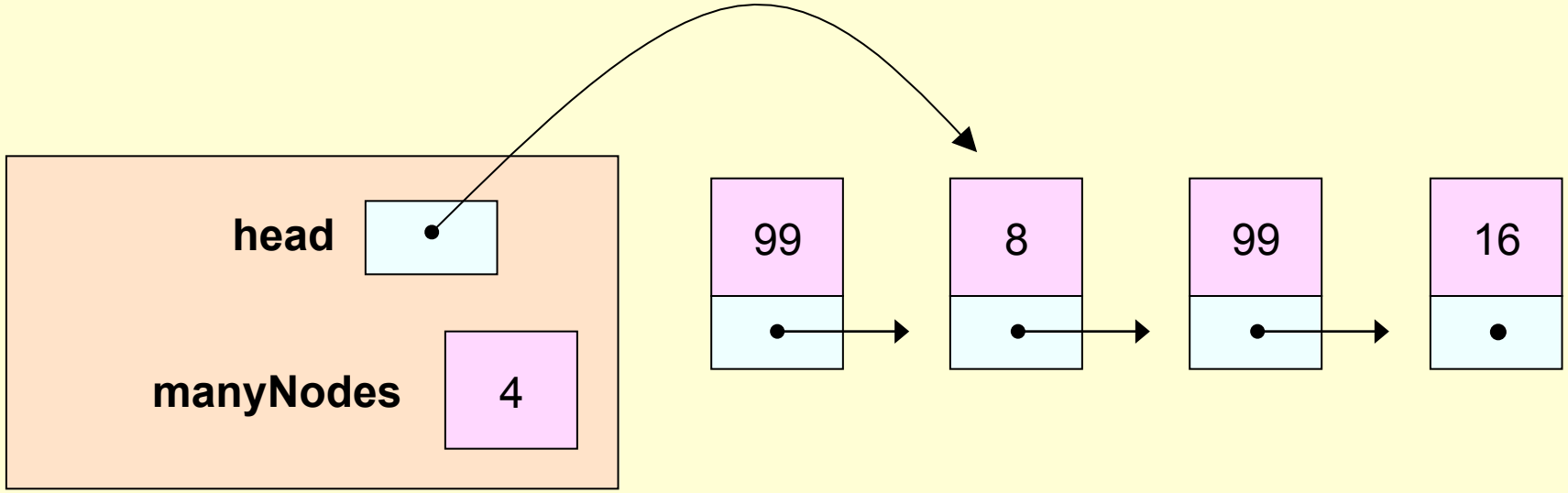
```
targetNode.setData(head.getData());
```



The remove method (3)

Now remove the head node to get rid of one of the nodes containing 99. Since the bag is not ordered it doesn't matter that this changes the order of nodes in the list

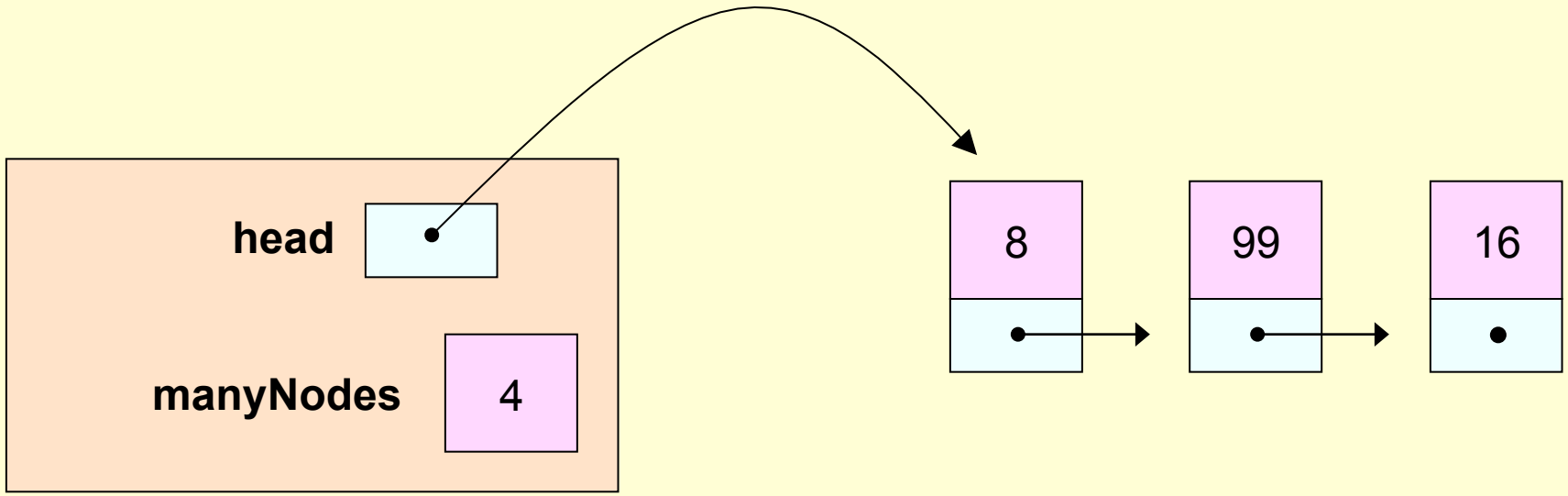
```
head = head.getLink();
```

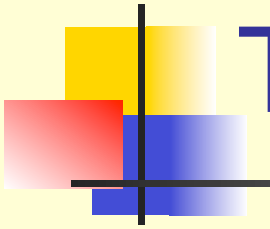


The remove method (4)

Now remove the head node to get rid of one of the nodes containing 99. Since the bag is not ordered it doesn't matter that this changes the order of nodes in the list

```
head = head.getLink();
```

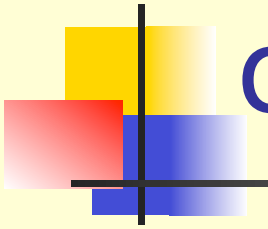




The remove method (5)

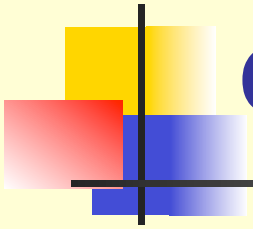
Remove a node from the bag list by copying the head data to the node to be removed and then remove the head. This works because there is no order on a bag

```
public boolean remove(int target)
{  IntNode targetNode =
                IntNode.listSearch(head, target);
   if (targetNode == null)
      return false; // target not found
   else
   {  targetNode.setData(head.getData());
      head = head.getLink(); // remove head
      manyNodes--;
      return true;
   }
}
```



countOccurrences method (1)

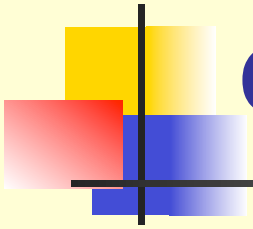
- There are two ways to do it
- Make a list traversal and count the number of times the given data element occurs
- Use the listSearch method in the IntNode class in a loop to continually search for the next occurrence of the data element.
- Main chooses to use listSearch in a loop



countOccurrences method (2)

Use the listSearch method from the IntNode class in a loop to count the number of occurrences of a given element.
This is Main's approach

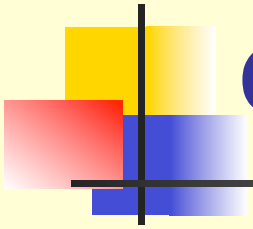
```
public int countOccurrences(int target)
{
    int answer = 0;
    IntNode cursor =
            IntNode.listSearch(head, target);
    while (cursor != null)
    {  answer++;
       cursor = cursor.getLink();
       cursor = IntNode.listSearch(cursor,target);
    }
    return answer;
}
```



countOccurrences method (3)

Here is an alternate method that doesn't use listSearch

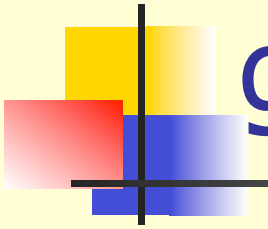
```
public int countOccurrences(int target)
{
   int answer = 0;
   IntNode cursor = head;
   while (cursor != null)
   {  if (target == cursor.getData())
         answer++;
      cursor = cursor.getLink();
   }
   return answer;
}
```



countOccurrences method (4)

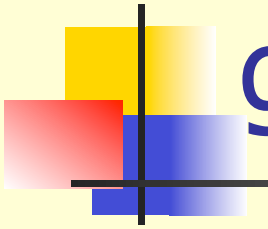
A similar method that uses a standard for loop

```
public int countOccurrences(int target)
{
   int answer = 0;
   for (IntNode cursor = head; current != null;
      cursor = cursor.getLink())
   {
      if (target == current.getData())
         answer++;
   }
   return answer;
}
```



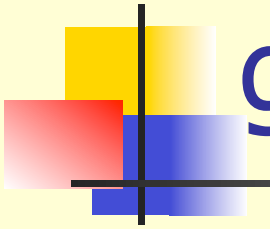
grab method (1)

- This method was not in IntArrayList
- It's purpose is to randomly select a data element from the bag and return it
- A random index can be generated using
  - `i = (int)(Math.random() * manyNodes) + 1; // i=1,2,3,...`
- Another way would have been
  - `Random random = new Random();`
  - `i = random.nextInt(manyNodes) + 1;`



grab method (2)

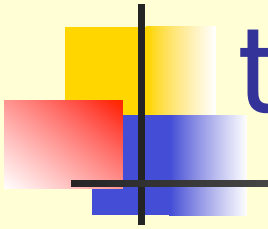
- There are two ways to find the list node at position i (i=1,2,3,...)
  - Use the listPosition method from the IntNode class
  - Just use an iterator that counts as elements are traversed and stops at ith node.
- Main uses the first approach. The listPosition method assumes that indices start at 1



grab method (3)

This method generates a random index in the range 1 to manyNodes. finds that node in the list using the listPosition method in the IntNode class, and returns the data value.

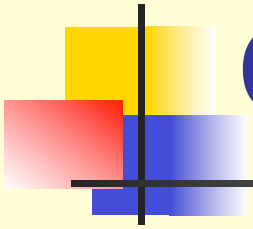
```
public int grab()
{
   if (manyNodes == 0)
   {
      throw new IllegalStateExeption("...");
   }
   int i = (int)(Math.random() * manyNodes) + 1;
   IntNode cursor =
                     IntNode.listPosition(head, i);
   return cursor.getData();
}
```



toString method

Main does not include a toString method.

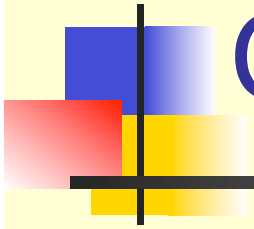
```
public int toString()
{
   StringBuffer s = new StringBuffer();
   s.append("IntLinkedBag[");
   IntNode current = head;
   while (current != null)
   {  s.append(current.getData());
      if (current.getLink() != null)
         s.append(",");
      current = current.getLink();
   }
   s.append("]");
   return s.toString();
}
```



Compare Bag implementations

Operation	array bag	linked bag
Constructor	O(capacity)	O(1)
add	O(1) / O(n)	O(1)
b1.addAll(b2)	O(n2) / O(n1+n2)	O(n2)
clone	O(capacity)	O(n)
countOccurrences	O(n)	O(n)
remove	O(n)	O(n)
ensureCapacity	O(capacity)	---
trimToSize	O(n)	---
union(b1,b2)	O(capacity1+capacity2)	O(n1+n2)
size	O(1)	O(1)

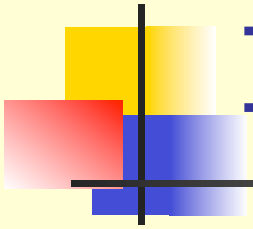
Since the bag is an unordered structure there is not much difference between the implementations. The main advantage is that we don't need to worry about the capacity for the linked implementation.



COSC 2006: Data Structures I

The IntLinkedSet class

Programming exercise



IntLinkedSet

- Modify the IntLinkedBag class to get an IntLinkedSet class
- We did this for IntArrayBag
- Make sure add and addAll don't allow duplicates to be added.
- Also replace countOccurrences by a method with prototype
  - `public boolean contains(int target);`