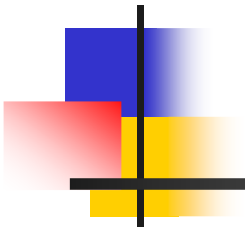
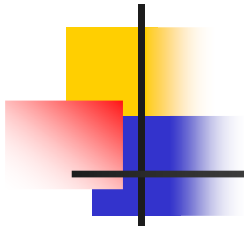


Ch6. Linear Lists – Linked Representation

© copyright 2006 SNU IDB Lab.

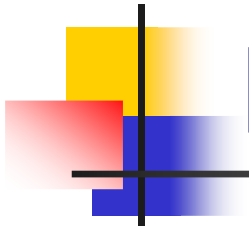
SNU
IDB Lab.



Bird's-Eye View

- Ch.5 ~ Ch.7: Linear List
  - Ch. 5 – array representation
  - Ch. 6 – linked representation
    - Singly Linked Lists and Chains
    - Circular Lists and Header Nodes
    - Doubly Linked Lists
    - Applications
  - Ch. 7 – simulated pointer representation

  ※ In succeeding chapters - matrices, stacks, queues, dictionaries, priority queues
- Java's linear list classes
  - java.util.ArrayList, java.util.Vector, java.util.LinkedList



Bird's-Eye View

- Linked representation of a linear list
  - Each element has an explicit pointer (reference in java)
- Data structure concepts introduced in this chapter
  - Linked representation
  - Chains, circular lists, and doubly linked lists
  - Header nodes
- java.util.LinkedList
  - Implements a linear list using pointers
  - Uses a doubly linked list
- Applications
  - Bin sort (bucket sort), radix sort – linear time sorting
  - Convex hull

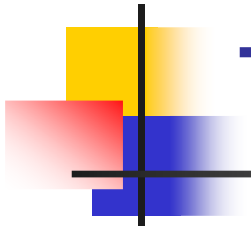
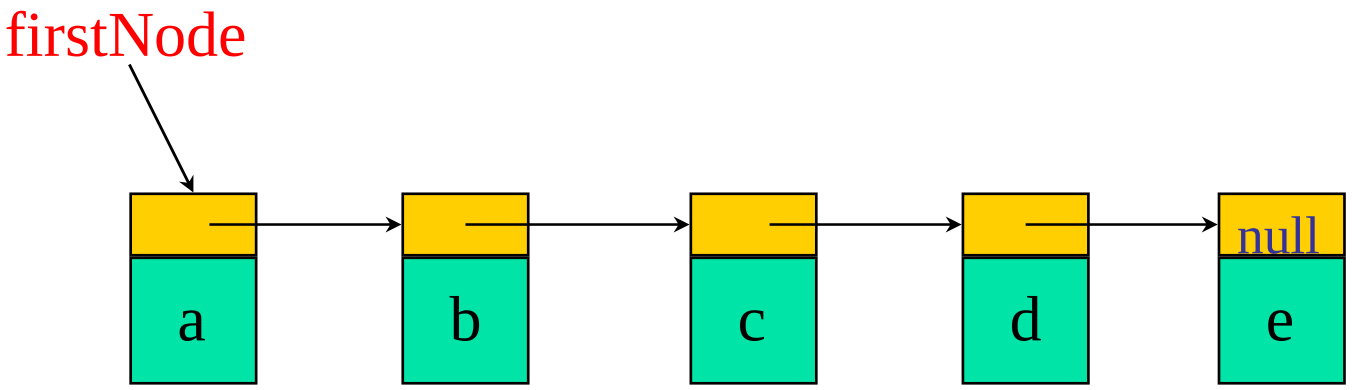


Table of Contents

- Singly Linked Lists and Chains
- Circular Lists and Header Nodes
- Doubly Linked Lists
- Applications

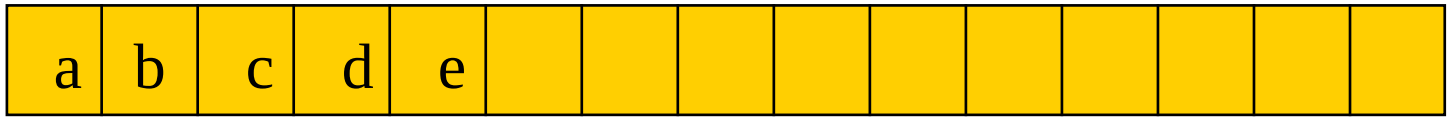
Linked Representation

- Each element is represented in a cell or node
- Elements are stored, in memory, in arbitrary order
- Explicit information (called link or pointer) is used to go from one element to the next element

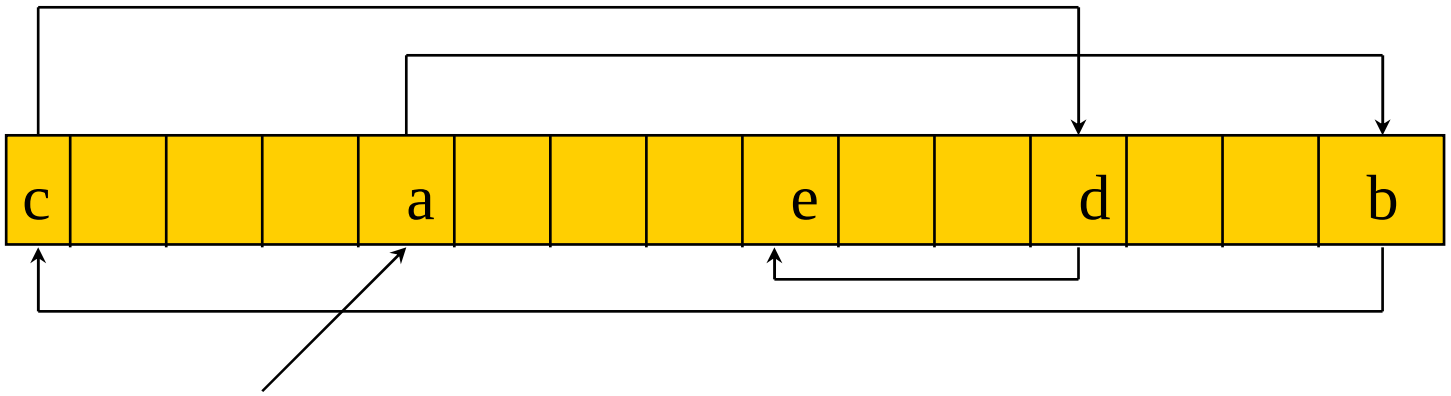


Memory Layout

- Layout of L = (a,b,c,d,e) using an array representation

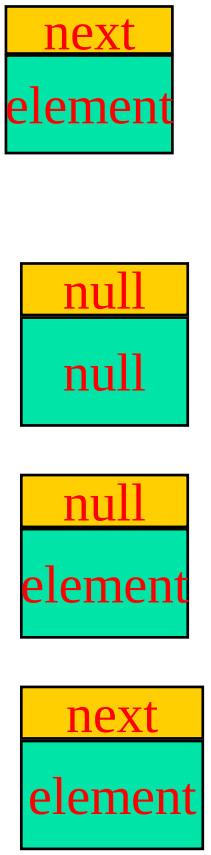


- A linked representation can have an arbitrary layout
  - Pointer in e is null
  - Use a variable firstNode to get the first element a

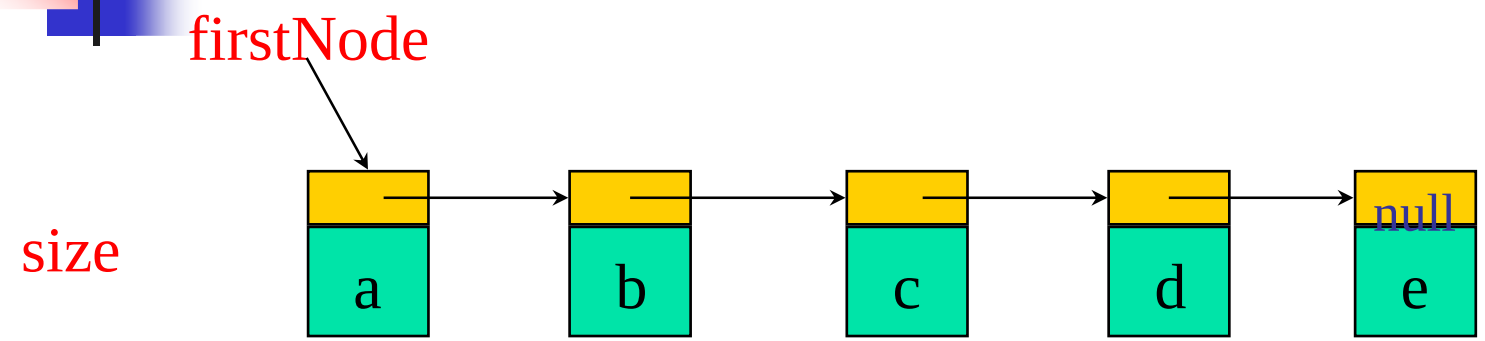


The class ChainNode

```
package dataStructures;
class ChainNode
{ // package visible data members
   Object element;
   ChainNode next;
  // constructors come here
   ChainNode() {}
   ChainNode(Object element}
      {this.element = element;}
   ChainNode(Object element, ChainNode next)
      {this.element = element;
       this.next = next;}
}
```

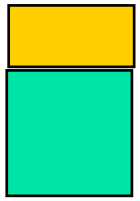


Chain (synonym for singly linked list)



- A chain is a linked list in which each node represents one element
- There is a link or pointer from one element to the next
- The last node has a null pointer
- Size: Number of elements

Use the class ChainNode



next (datatype ChainNode)

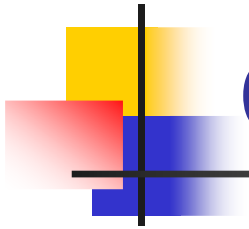
element (datatype Object)

The Class Chain : Implementation

```
package dataStructures;
import java.util.*;  // will use IndexOutOfBoundException
public class Chain implements LinearList // linked implementation
{  protected ChainNode firstNode;
   protected int size;
   // constructors
   public Chain(int initialCapacity)
   { // the default initial values of firstNode and size are null and 0, respectively.
     // do nothing  }
   public Chain()  { this(0); }
   // other methods of Chain come here…….
}
```

** In the beginning, we just create an empty chain

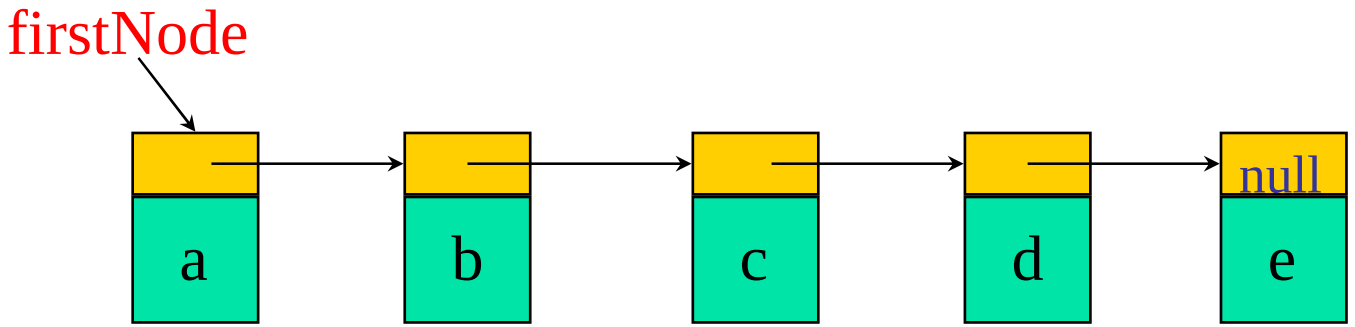
ex) studentlitst = new Chain(5); (equivalent) studentlist = new Chain();



Chain Methods

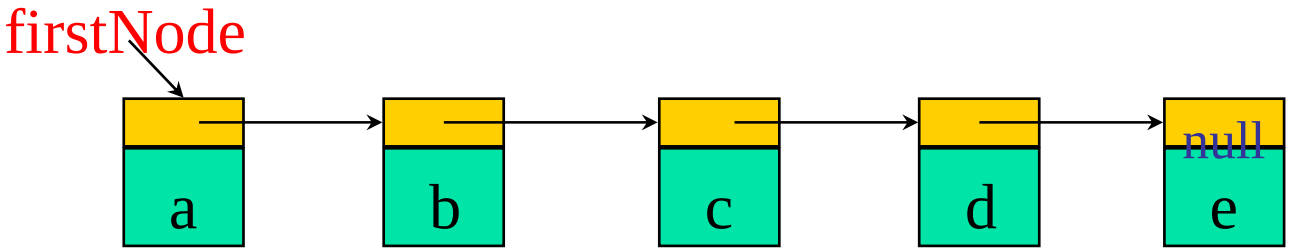
```
/** @return true iff list is empty */
  public boolean isEmpty()     {return size == 0;}

/** @return current number of elements in list */
  public int size()    {return size;}

/** @throws IndexOutOfBoundsException when index is not between 0 and size - 1 */
  void checkIndex(int index)
  {  if (index < 0 || index >= size)
       throw new IndexOutOfBoundsException  ("index = " + index + "  size = " + size);
   }
```

Chain Method: get()



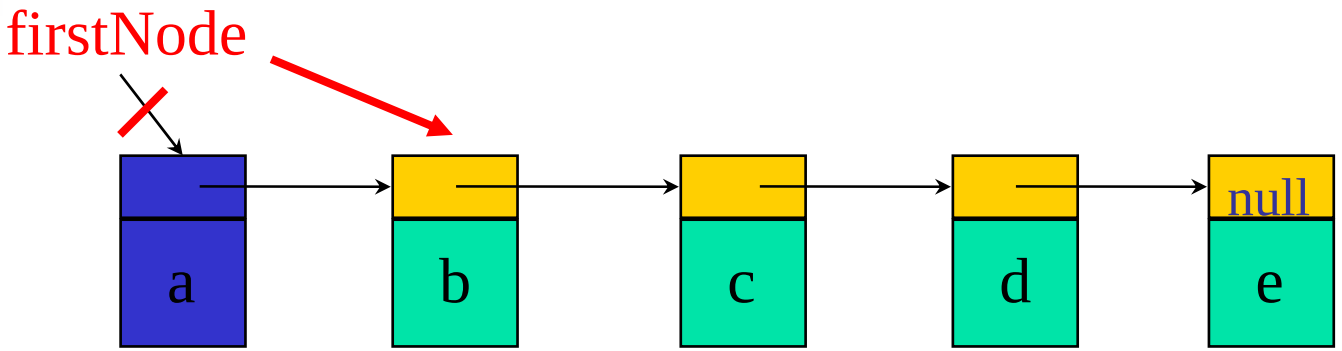
- get(0)
  - desiredNode = firstNode; // gets you to first node
- get(1)
  - desiredNode = firstNode.next; // gets you to second node
- get(2)
  - desiredNode = firstNode.next.next; // gets you to third node
- get(6) throws NullPointerException
  - desiredNode = firstNode.next.next.next.next.next.next

Code for get()



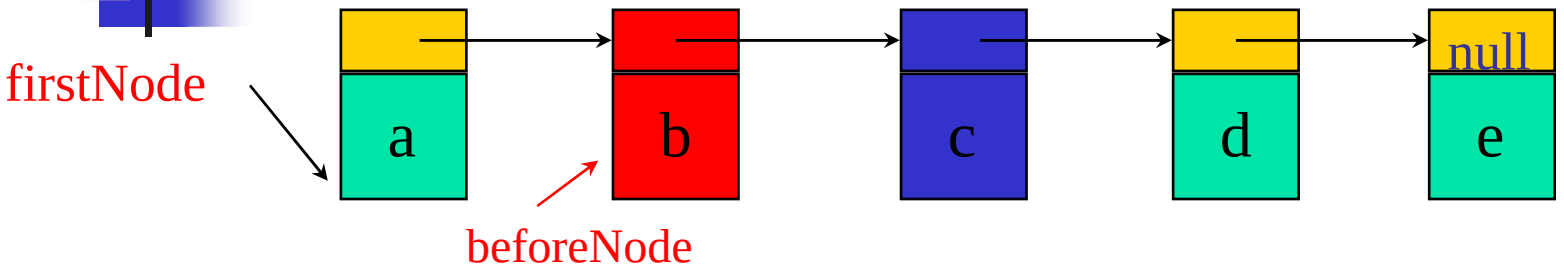

```
public Object get (int index)
{     checkIndex(index);
      // move to desired node
      ChainNode currentNode = firstNode;
      for (int i = 0; i < index; i++)
                  currentNode = currentNode.next;
      return currentNode.element;
}
```

Chain method remove(): 1st node case

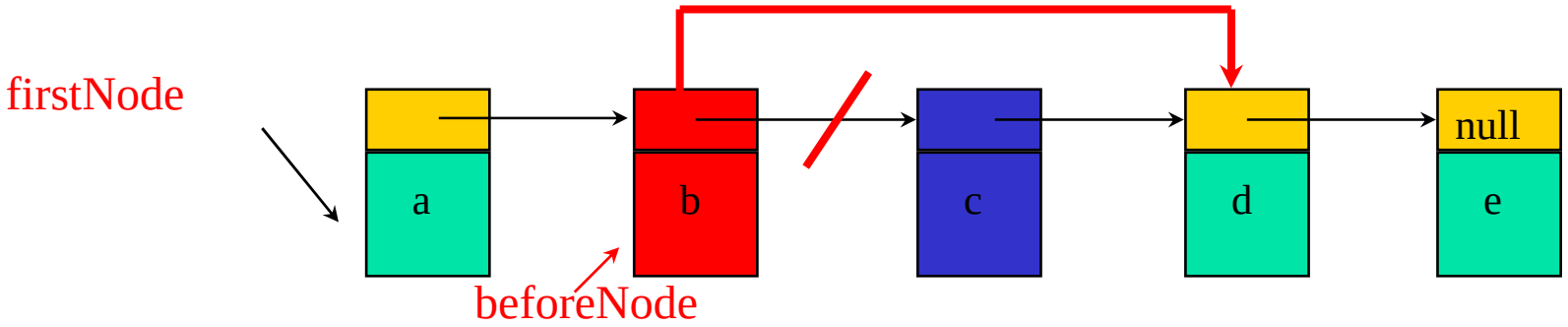


- remove(0)
- firstNode = firstNode.next

Chain method remove(): other cases



- First get to node just before node to be removed
- beforeNode = firstNode.next



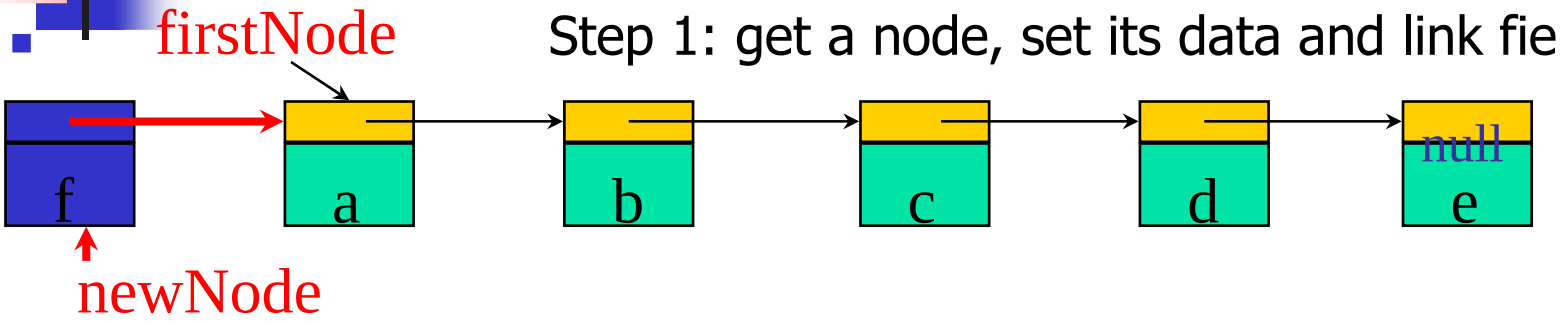
- Now change pointer in beforeNode
- beforeNode.next = beforeNode.next.next

Code for remove()

```
public Object remove(int index)
{ checkIndex(index);
   Object removedElement;
   if (index == 0) // remove first node
      {  removedElement = firstNode.element;
         firstNode = firstNode.next; }
   else  {  // use q to get to predecessor of desired node
         ChainNode q = firstNode;
         for (int i = 0; i < index - 1; i++)          q = q.next;
         removedElement = q.next.element;
         q.next = q.next.next; // remove desired node }
   size--;
   return removedElement;
}
```

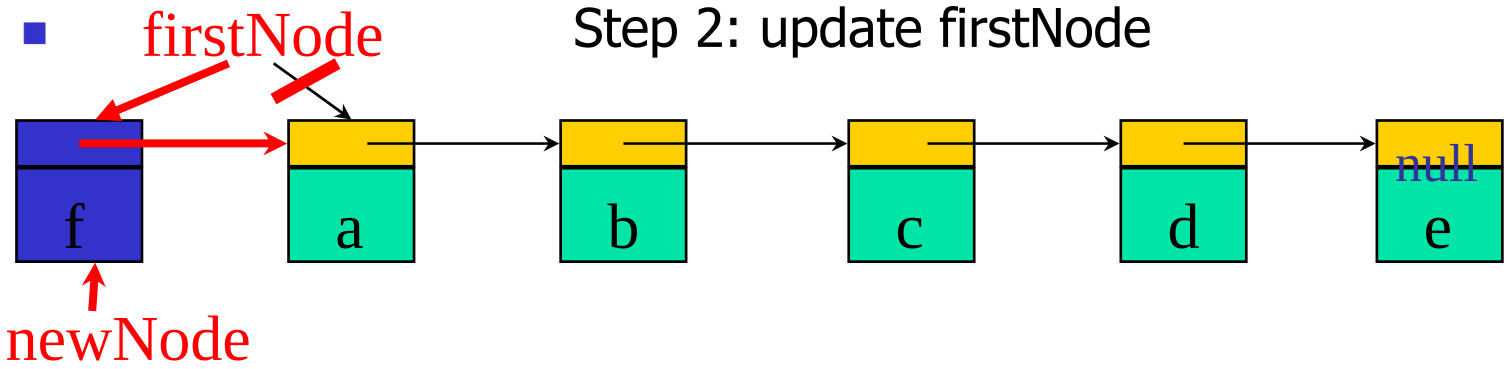
Chain method add(0,'f') --at front by 2 steps

- Step 1: get a node, set its data and link fields



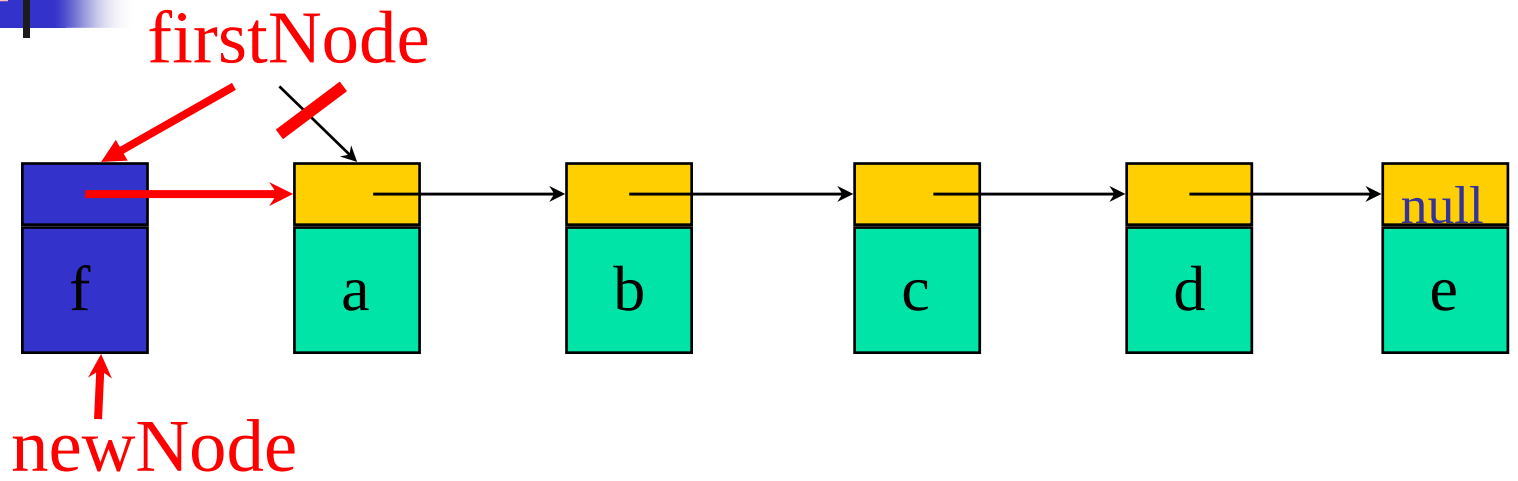
- ChainNode newNode = new ChainNode(new Character('f'), firstNode);

- Step 2: update firstNode



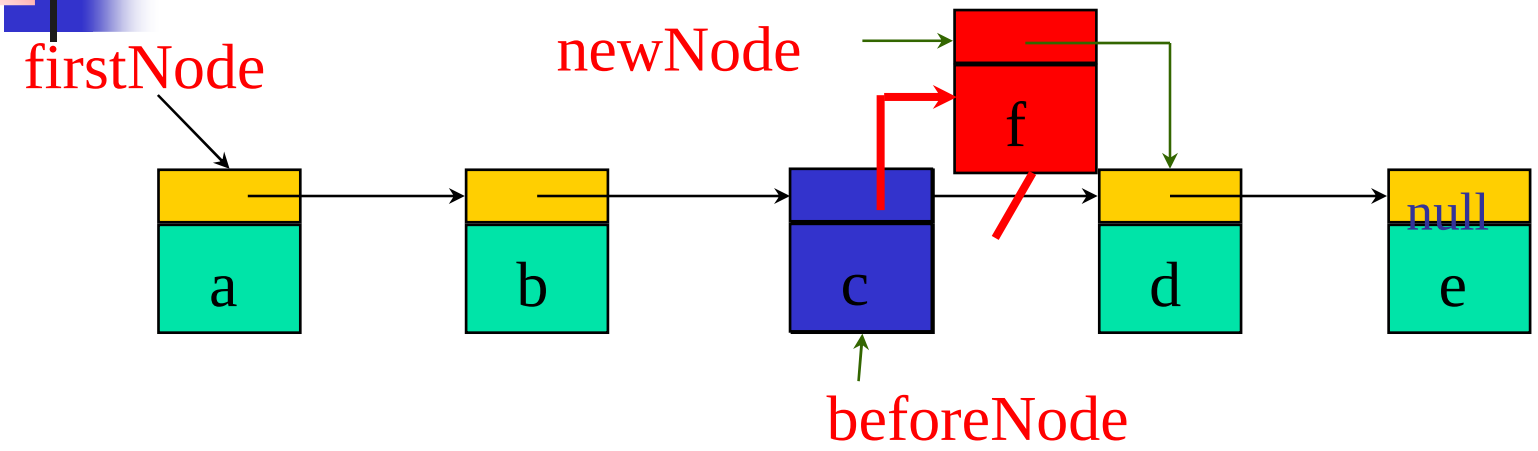
- firstNode = newNode;

Chain method add(0, 'f') -- at front by 1 step



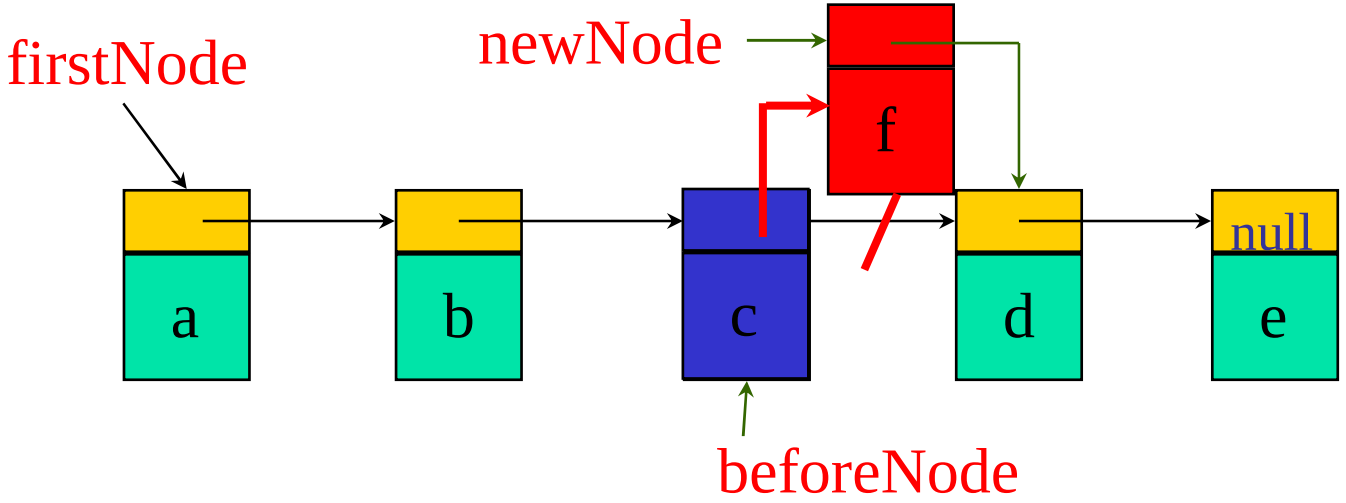
- firstNode = new ChainNode(new Character('f'), firstNode);

Chain method add(3, 'f') -- not front case by 3 steps

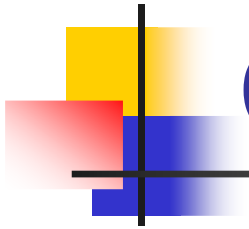


- First find node whose index is 2
  - beforeNode = firstNode.next.next
- Next create a node and set its data and link fields
  - ChainNode newNode = new ChinNode(new Character('f'), beforeNode.next);
- Finally link beforeNode to newNode
  - beforeNode.next = newNode;

Chain method add(3, 'f') -- not front case by 2 steps



- beforeNode = firstNode.next.next; // find beforeNode
- beforeNode.next = new ChainNode(new Character('f'), beforeNode.next);



Code for add()

```
public void add(int index, Object theElement)
 { if (index < 0 || index > size) // invalid list position
        throw new IndexOutOfBoundsException ("index = " + index + "  size = " + size);
   if (index == 0)  // insert at front
       firstNode = new ChainNode(theElement, firstNode);
    else  { // find predecessor of new element "beforeNode"
      ChainNode p = firstNode;
      for (int i = 0; i < index - 1; i++)     p = p.next;
      // insert after p "beforeNode"
      p.next = new ChainNode(theElement, p.next); }
   size++;
 }
```

Performance

Operation	FastArrayLinearList	Chain	IAVL
get	5.6ms	157sec	63 ms
best-case adds	31.2ms	304ms	253 ms
average adds	5.8sec	115sec	392 ms
worst-case adds	11.8sec	157sec	544 ms
best-case removes	8.6ms	13.2ms	1.3 sec
average removes	5.8sec	149sec	1.5 sec
worst-case removes	11.7sec	157sec	1.6 sec

- FastArrayLinearList(ch5) showed better results
  - Array-based class used a finely tuned array copy method provided by java (System.arraycopy method)
- IAVL: Indexed AVL Tree

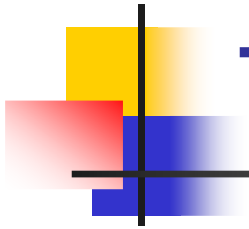
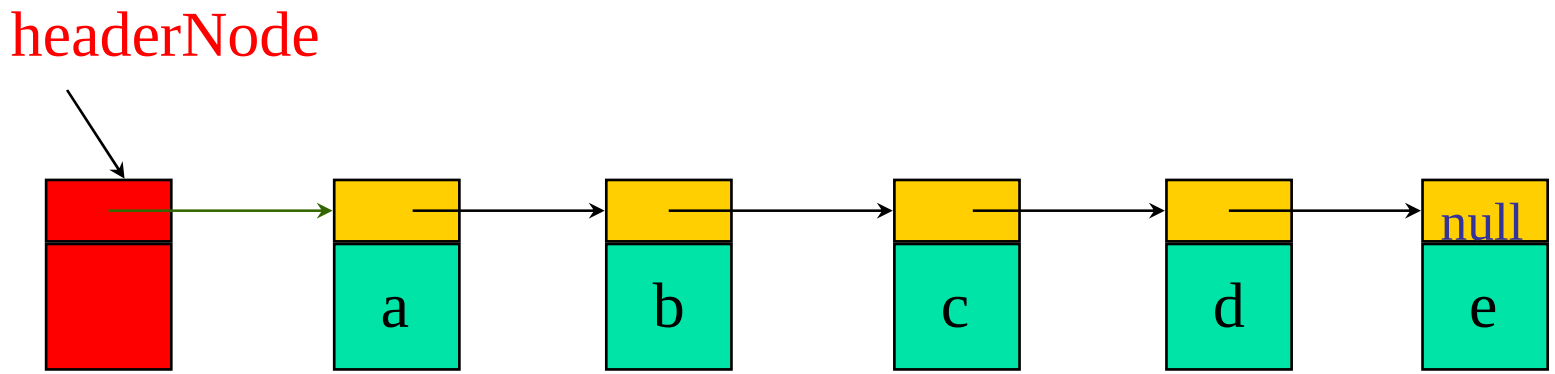


Table of Contents

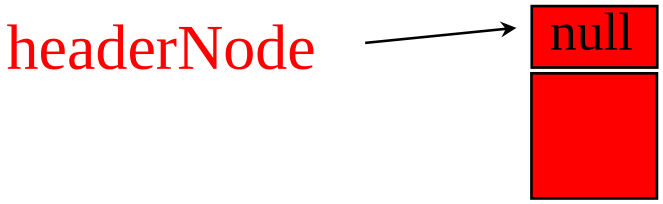
- Singly Linked Lists and Chains
- Circular Lists and Header Nodes
- Doubly Linked Lists
- Applications

Chain with Header Node

- Add an additional node (header node) at the front



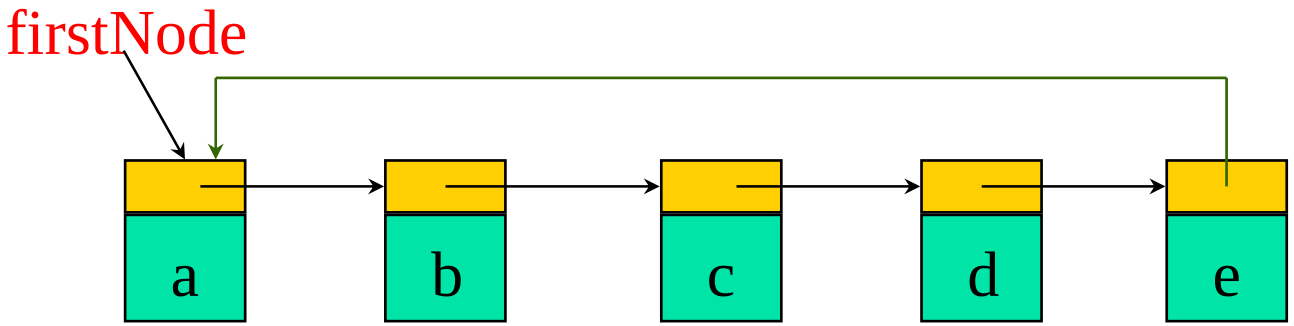
- Empty Chain with Header Node



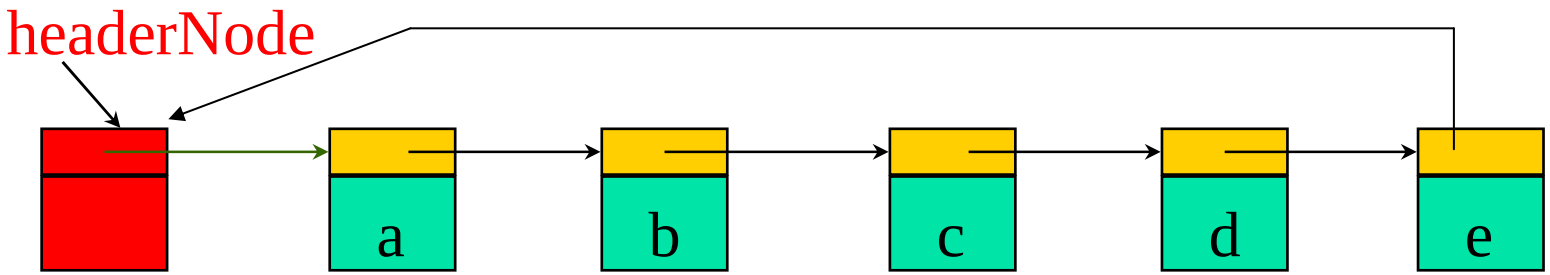
- The node space of header node can be used effectively for saving extra information

Circular List

- Without header



- With header



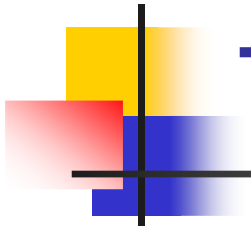
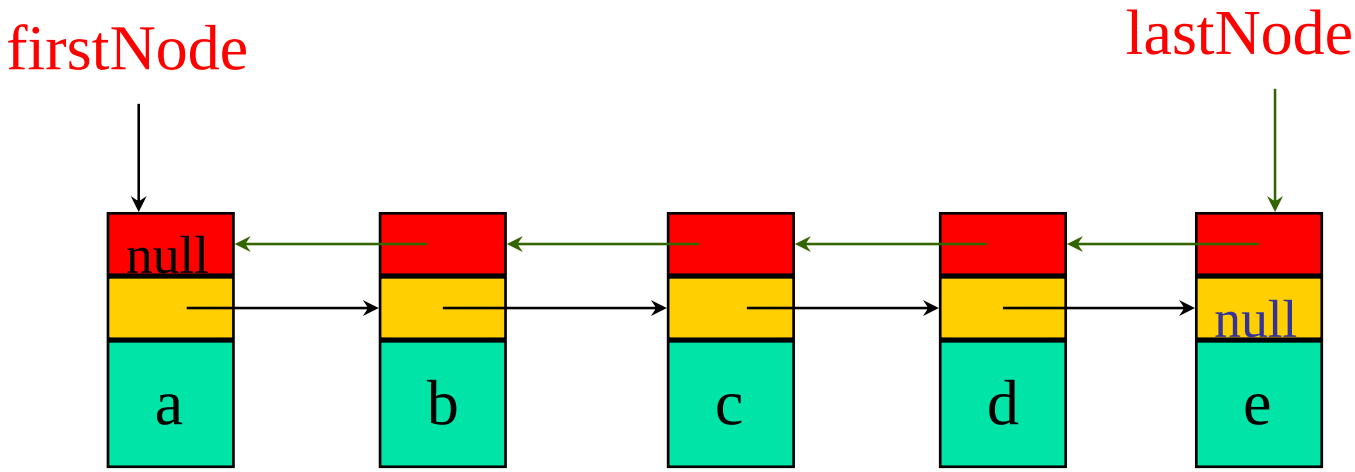


Table of Contents

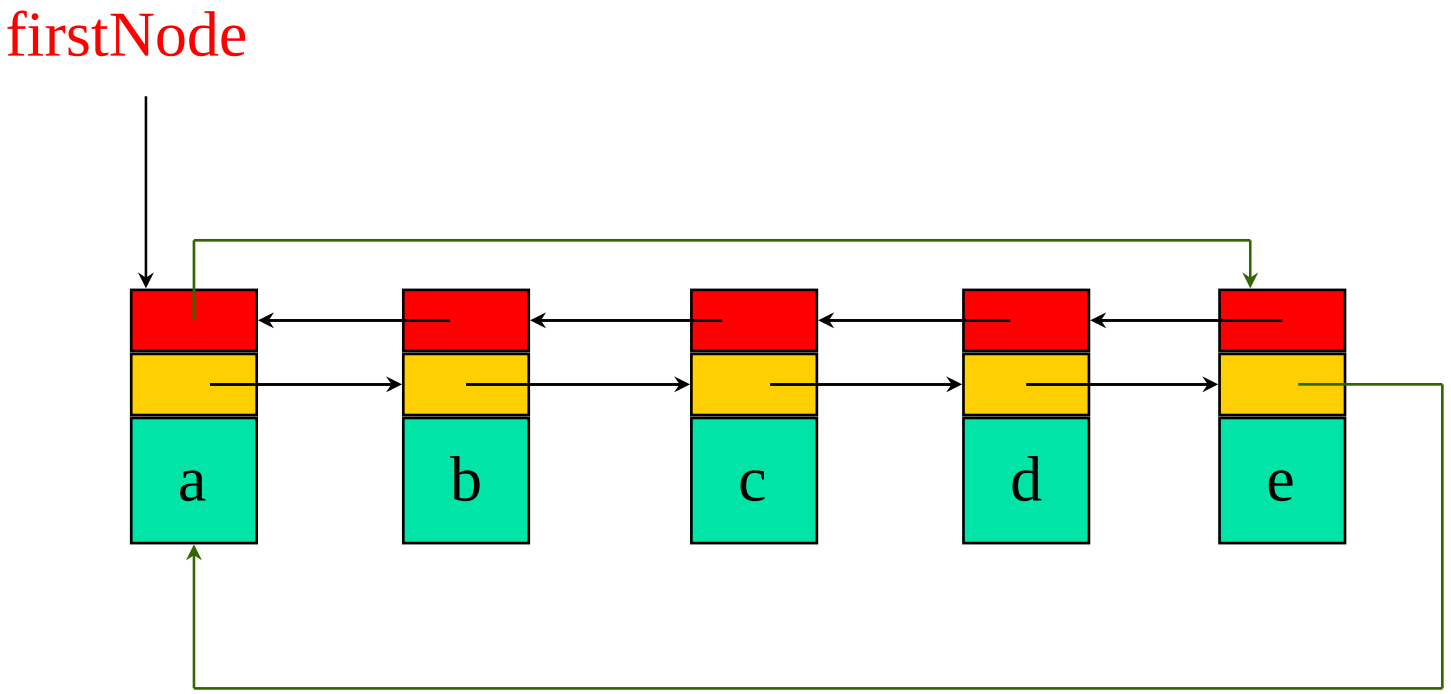
- Singly Linked Lists and Chains
- Circular Lists and Header Nodes
- Doubly Linked Lists
- Applications

Doubly Linked Lists

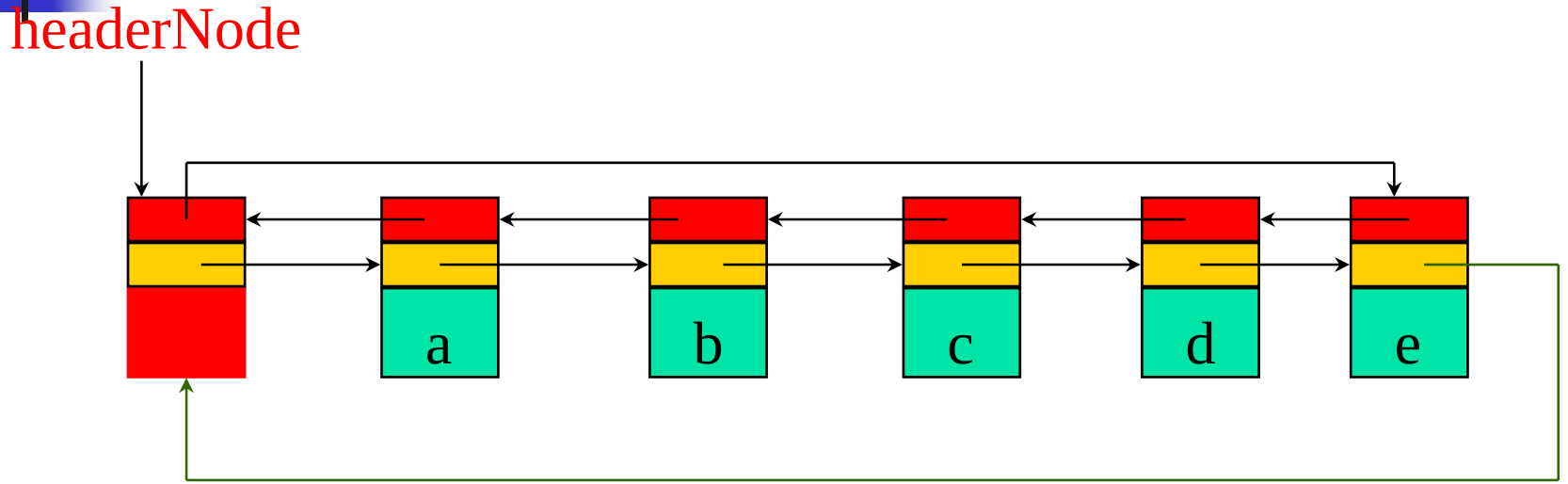


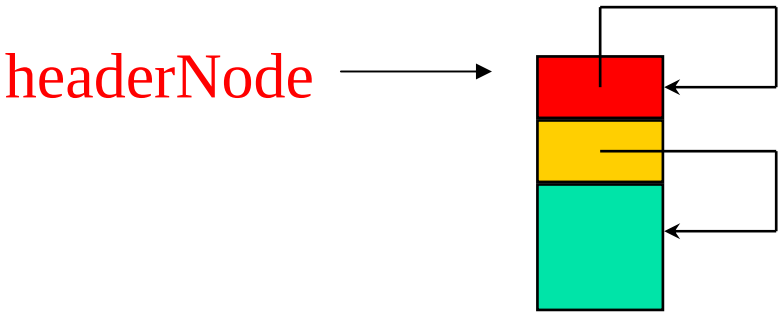
- An ordered sequence of nodes in which each node has two pointers: next and previous
- Two instance data members: firstNode and lastNode
- Not circular yet!

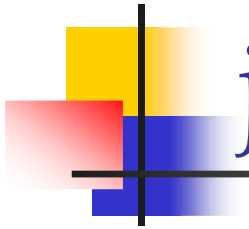
Doubly Linked Circular List(DLCL)



DLCL with Header Node







java.util.LinkedList

- Java Linked implementation of linear list
- Doubly linked circular list with header node!
- Has all methods of LinearList
  - isEmpty, size, get, indexOf, remove, ass, toString
  - plus many more (ex: array to linked list conversion, etc)

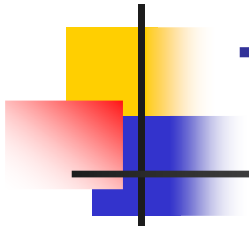
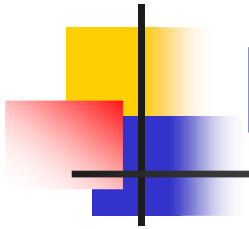


Table of Contents

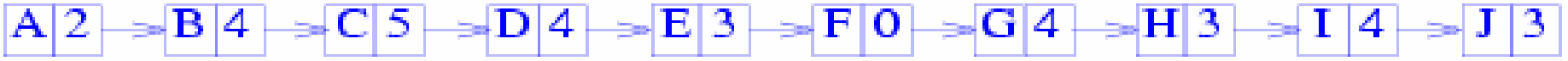
- Singly Linked Lists And Chains
- Circular Lists And Header Nodes
- Doubly Linked Lists
- Applications
  - Bin Sort: linear list application
  - Convex Hull: doubly linked list application



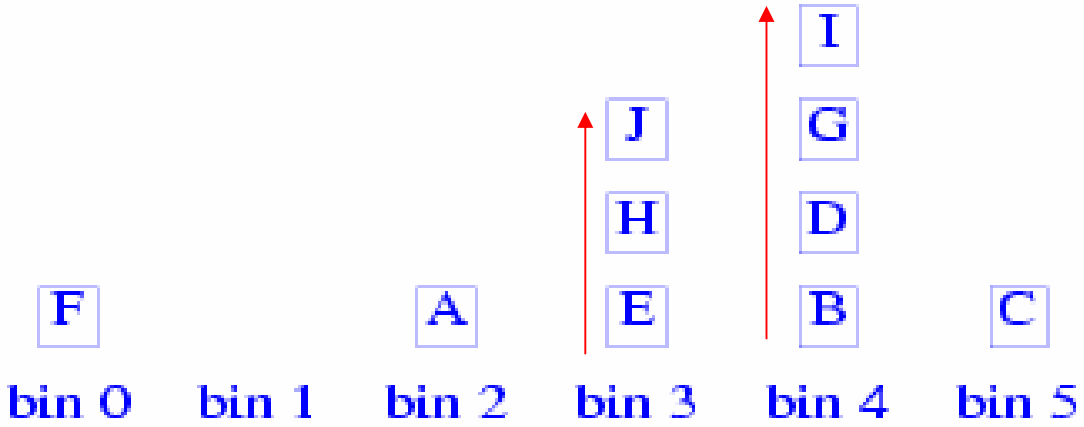
Bin Sort

- Motivation
  - A chain is used to maintain a list of students (40 students) in a class
  - All scores are integers in a range (say, 0 through 100)
  - A node has fields for name, score, address, etc
  - Want to sort the nodes in order of the score
- Sorting nodes with bin sort
  - Nodes are places into bins
  - Each bin containing nodes with the same score
  - Combine the bins to create a sorted chain
  - Possible only when having the closed range of discrete values!

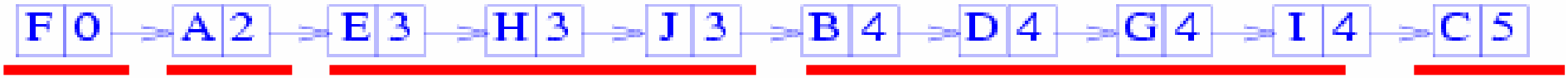
Bin Sort Example (1)



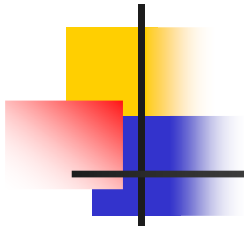
(a) Input chain



(b) Nodes in bins



(c) Sorted chain

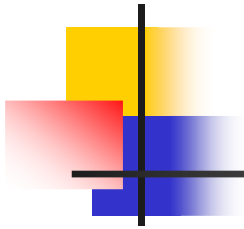


Bin Sort Example (2)

- Assume that each name is a single character
- Scores are in the range 0 through 5
  - Need six bins (each bin is a linked list! and for each number)
- Steps in bin sort
  - Initialize bins & Examine the nodes one at a time
  - Examined nodes are placed into the corresponding bin
  - Collect the nodes from the bins, beginning with those in bin 0
- Complexity
  - No. of operations for examining nodes in a chain and nodes in bins
- The smaller # of bins, the better performance!

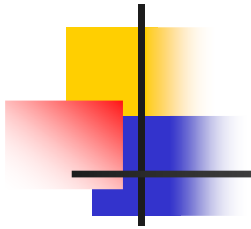
The Class ScoreObject and StudentRecord

```
public class ScoreObject {
      int score;
      public SroreObject(int theScore)
         { score = theScore;}
}

public class StudentRecord extends ScoreObject {
      String name;
      public StudentRecord(String Name, int Score) {
         super(Score);
         name = Name;
      }
      ….
}
```



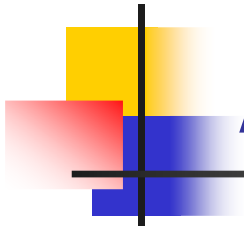
Bin sort using the methods of Chain

```
Public static void binSort (Chain theChain, int range)  { //sort by score
  Chain[] bin = new Chain[range + 1];                      // array declaration
  for (int i =0; i<=range; i++)   bin[i] = new Chain(); // array initialization

//distribute the elements from the Chain to bins
 int numberOfElements = theChain.size();
 for (int i=1; i<= numberOfElements; i++) {
      ScoreObject theElement = (ScoreObject) theChain.remove(0);
      bin[theElement.score].add(0, theElement); }

// collect elements from bins
 for (int j = range; j >= 0; j--)  while (!bin[j].isEmpty()) {
        ScoreObject theElement = (ScoreObject) bin[j].remove(0);
        theChain.add(0, theElement); }
}
```



Bin Sort: working mechanism

- Our concern
  - number of operations for examining nodes in a chain or nodes in bins
- You cannot do anything for the step 2
- However in step 3
  - The 2nd for loop examines the bins beginning with 0 and concatenates the chains in the nonempty bins
  - If too many empty bins, not desirable!
  - We want to reduce empty bins!



Analysis of Bin Sort

- Overall Complexity: $O(n + \text{range})$
  - n: number of items to sort
  - range: the range of items to sort which is the # of bins
  - To sort *n* integers with the range 0 through $n^c - 1$
    - $\Theta(n + \text{range}) = \Theta(n^c)$ (at least c >= 1)
    - 340 items with range of 1—999 ➔ $n^c = 340^c = 999$

- The property of "stable sort" is important
  - Does not change the relative order of elements that have same score within a bin during the binSort

Radix Sort

- Pitfall of Bin Sort
  - $\Theta(n + range) = \Theta(n^c)$
  - When the range is big (too many bins!), Bin sort is not useful!

- Radix Sort
  - Decompose the numbers into digits using some radix r and then sort them by digits ➔ $\Theta(n)$
  - Downsize the range (# of bins) using radix
    - 928 = 9*10*10 + 2*10 + 8 (by radix 10)
    - 3275 = 1*60*60 + 2*60 + 5 (by radix 60)

Radix Sort Example

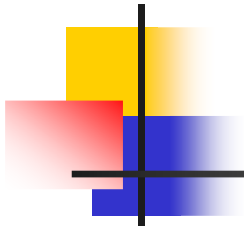
(a) Input chain

(b) Chain after sorting on least significant digit

(c) Chain after sorting on second-least significant digit

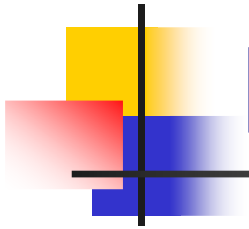
(d) Chain after sorting on most significant digit

- Apply Bin Sort from (a) to (b), from (b) to (c), and from (c) to (d)!
  - ONLY NEED 10 BINS IN EACH STEP
  - The property of “stable sort” is important here!



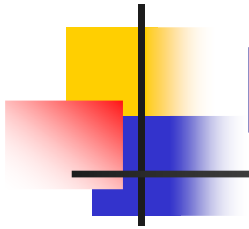
Radix sort example

- Input: 216, 521, 425, 116, 96,515, 124, 34, 96, 24
- 1의 자리 bins
  - (521,91)( )( )( )(124,34,24)(425,515)(216,116,96)( )( )( )
- 10의 자리 bins (bin 내의 숫자들의 1의자리가 sorted!)
  - (515,216,116)(521,124,24,425)(34)( )( )( )( )(91,96)
- 100의 자리 bins (bin 내의 숫자들의 10의자리가 sorted!)
  - (24,34,86,91)(116,124)(216)( )(425)(515,521)( )( )( )( )



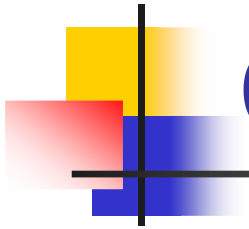
Radix sort complexity

- By bin sort with n = 10 numbers and range 1~999
  - Bin initialization (1000 steps), node distribution (10 steps), collection from bins (1000 steps) ➔ total 2010 steps
  - Complexity = $\Theta(n + range) = \Theta(n^c) = \Theta(10^3)$
- By radix sort using radix = 10 (range 0~9)
  - Building a sorted chain 3 times using 3 bin sorts
  - Bin initialization(10 steps), node distribution(10 steps), collection from bins (10 steps) ➔ 3 X 30 = total 90 steps
  - Each sorted chain building requires r(=n) operations
    - $\Theta(3*r) \rightarrow \Theta(3*n) \rightarrow \Theta(n)$



Non-empty bins

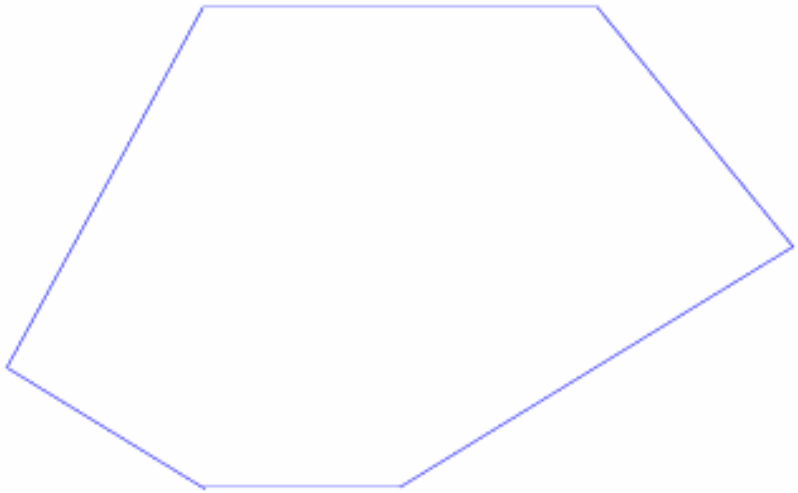
- Can we have only non-empty bins?
- Ex. 216, 521, 425, 116, 91, 515, 124, 34, 96, 24
  - Grouping better than radix sort (?)
    - Bin216, Bin521, Bin425…..
    - Bin24, Bin34, ……
- Oops! We are doing a great amount of comparisons ➔ another sorting!



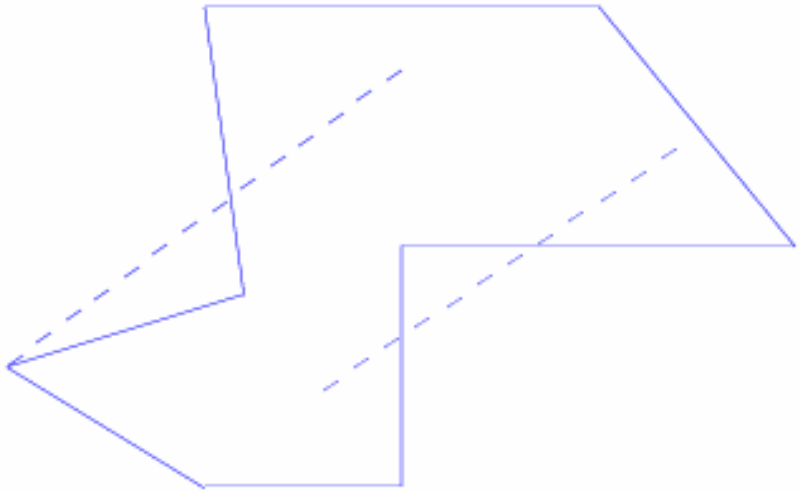
Convex Hull

- Polygon
  - A closed planar figure with three or more straight edges
  - A polygon contains all points that are either on its edges or inside the region it encloses
  - Representation: a doubly linked circular list of points
- Convex
  - A polygon is convex iff all line segments that join two points on or in the polygon include no point that is outside the polygon
- Convex Hull of a set *S* of points
  - The smallest convex polygon that contains all these points
  - The vertices (corners) of this polygon are the extreme points of *S*

Convex and Nonconvex Polygons

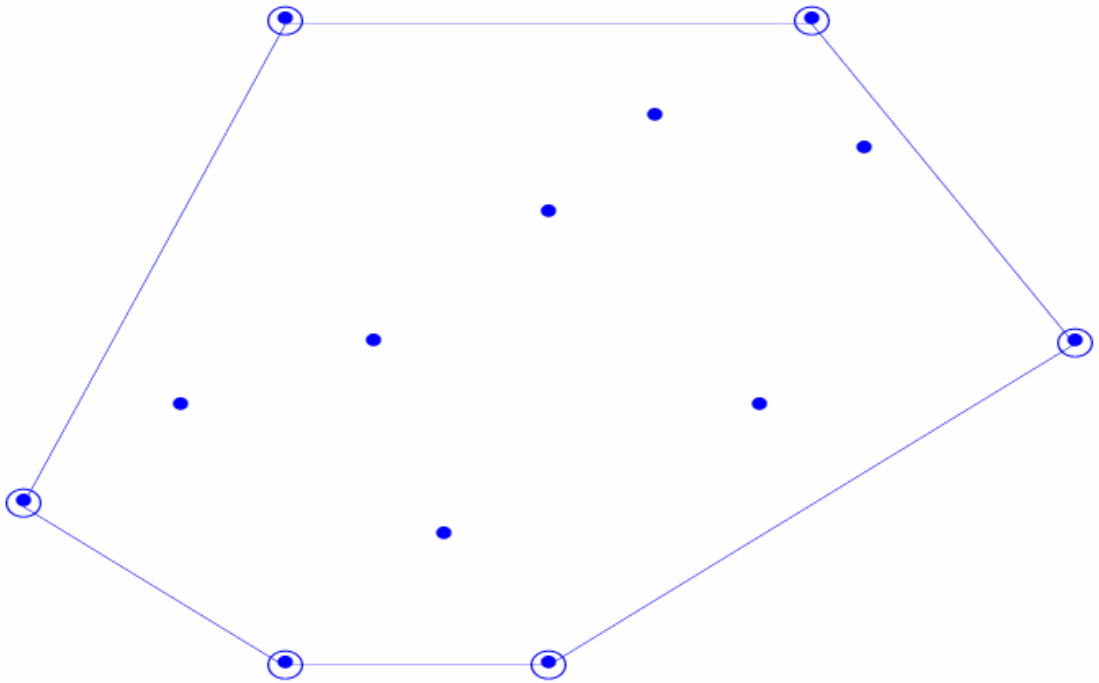


(a) Convex polygon



(b) Nonconvex polygon

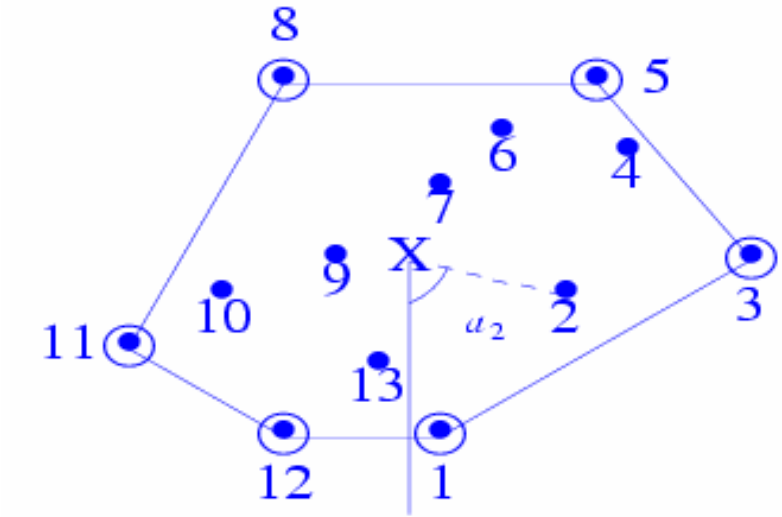
Convex Hull of Planar Points



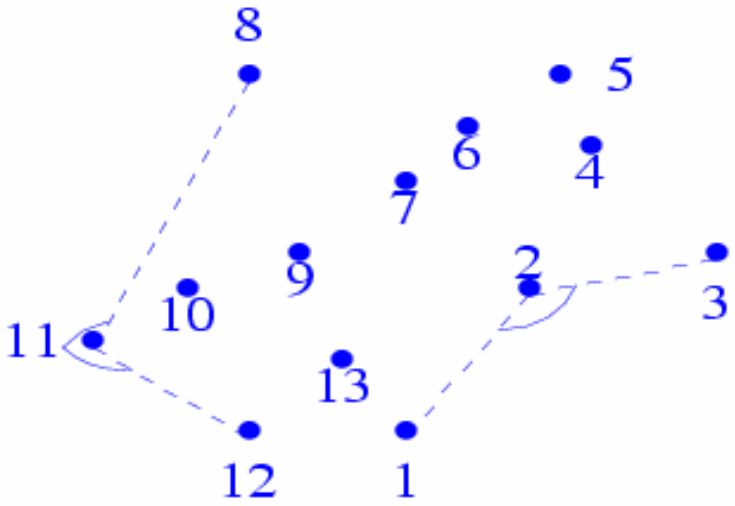
• ... Planar point

○ ... Extreme point

Identifying Extreme Points



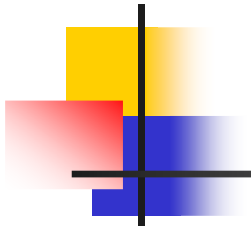
(a) Point ordering



(b) Counterclockwise angles

- Fix the center point X
- Sort the points by polar angle

- try (1,2,3): drop 2 & try (13,1,3): ok
- try (1,3,4): ok
- try (3,4,5): drop 4 & try (1,3,5): ok
- try (3,5,6) …..



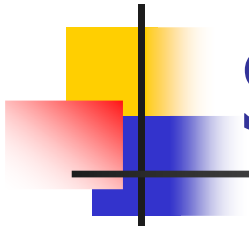
Pseudocode: Convex Hull of S (1)

- Step 1 [Handle degenerate cases]
  - If S has fewer than three points, return S
  - If all points lie on a straight line, compute the endpoints of the smallest line that includes all points of S and return these two points
- Step 2 [Sort by polar angle]
  - Find a point X that is inside the convex hull of S
    - ((sum of x coordinates)/ n, (sum of y coordinates)/n)
  - Sort S by polar angle and within polar angle by distance from X
  - Create a doubly linked circular list of points using above order
  - Let right link to the next point in the order and left link to the previous point

Pseudocode: Convex Hull of S (2)

- Step 3 [Eliminate nonextreme points]
  - Let p be the point that the smallest y-coordinate (break a tie, if any, by selecting the one with largest x-coordinate)

```
for (x = p; rx = point to the right of x; x != rx) {
    rrx = point to the right of rx;
    if (angle formed by x, rx, and rrx is <=180 degrees) {
        delete rx from the list;
        rx = x;
        x = point on left of rx;
    }
    else {
    x = rx; rx = rrx;}
}
```



Summary

- Linked representation of a linear list
  - Each element has an explicit pointer (reference in java)
- Data structure concepts
  - Linked representation
  - Chains, circular lists, and doubly linked lists
  - Header nodes
- java.util.LinkedList
  - Implements a linear list using pointers
  - Uses a doubly linked list
- Applications
  - Bin sort (bucket sort), radix sort – linear time sorting
  - Convex hull