CPSC 319 Tutorial 07
Single Linked List

Longsheng Zhou

Department of Computer Science
University of Calgary

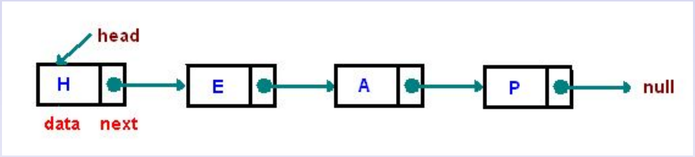
March 2, 2015

- **What is the disadvantage of using array?**[exercise]

- One disadvantage of using arrays to store data is that arrays are *static* structures and therefore cannot be easily *extended* or *reduced* to fit the data set.

- Arrays are also expensive to maintain new *insertions* and *deletions*.

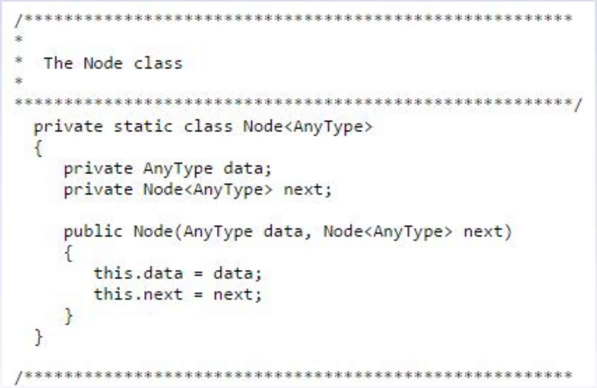
- A linked list is a linear data structure where each element is a separate object.



- Each element (we will call it a ***node***) of a list is comprising of two items:
  - Data: **data**
  - Reference to the next node: **next**

- The last node has a reference to *null*. The entry point into a linked list is called the *head* of the list. It should be noted that head is not a separate node, but the reference to the first node. If the list is empty then the head is a null reference.

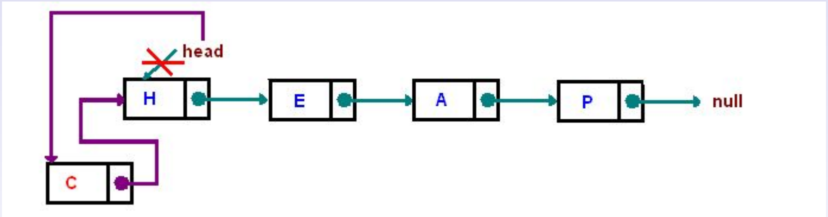
In Java you are allowed to define a class (inner class) inside of another class (outer class). Here is the java implementation of node class:



```
/*******************************************************
*
*  The Node class
*
********************************************************/
  private static class Node<AnyType>
  {
     private AnyType data;
     private Node<AnyType> next;

     public Node(AnyType data, Node<AnyType> next)
     {
        this.data = data;
        this.next = next;
     }
  }

/*******************************************************
```

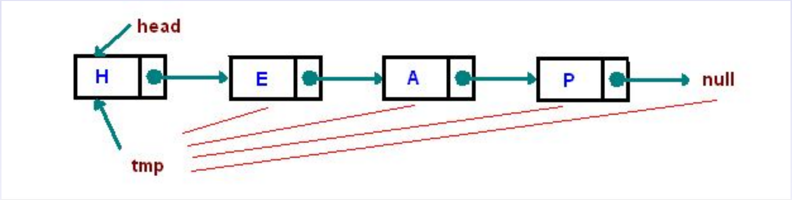
1. Add (addFirst, addLast)
2. Traverse
3. Insert
4. Delete

- **addFirst**
  The method creates a node and prepends it at the beginning of the list.



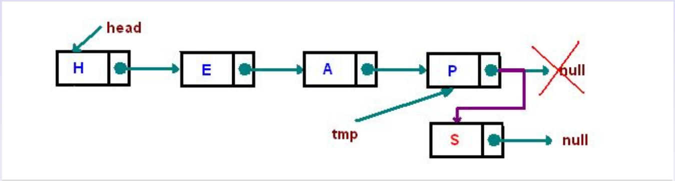
```
public void addFirst(AnyType item)
{
   head = new Node<AnyType>(item, head);
}
```

- **Traversing**
  Start with the head and access each node until you reach null. Do not change the head reference.



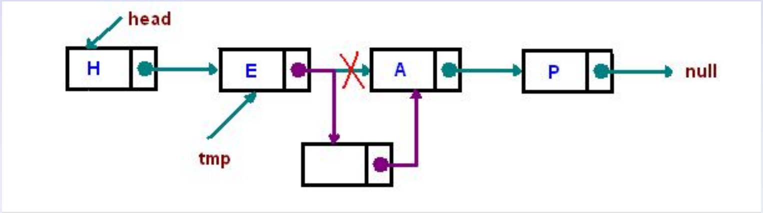
```
Node tmp = head;

while(tmp != null) tmp = tmp.next;
```

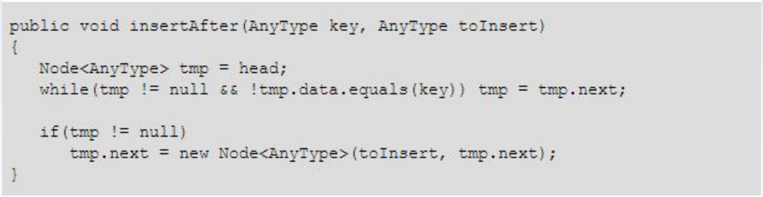
- **addLast**
  The method appends the node to the end of the list. This requires traversing, but make sure you stop at the last node.



```
public void addLast(AnyType item)
{
   if(head == null) addFirst(item);
   else
   {
      Node<AnyType> tmp = head;
      while(tmp.next != null) tmp = tmp.next;

      tmp.next = new Node<AnyType>(item, null);
   }
}
```

- **Inserting**
  Find a node containing “key” and insert a new node after it. In the picture below, we insert a new node after “E”:.



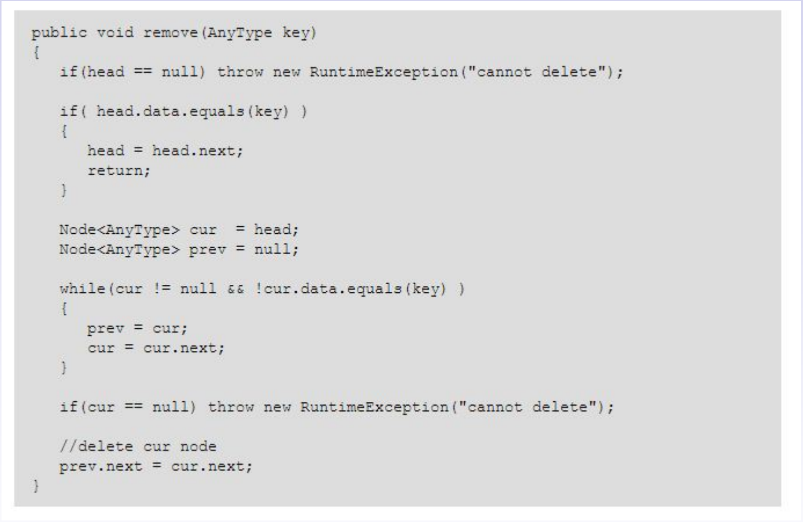


```
public void insertAfter(AnyType key, AnyType toInsert)
{
   Node<AnyType> tmp = head;
   while(tmp != null && !tmp.data.equals(key)) tmp = tmp.next;

   if(tmp != null)
      tmp.next = new Node<AnyType>(toInsert, tmp.next);
}
```

- **Delete**

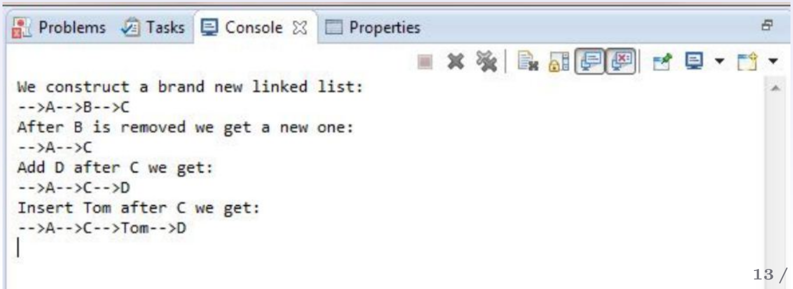
  Find a node containing “key” and delete it. In the picture below we delete a node containing ”A”. It is convenient to use two references *prev* and **cur**. When we move along the list we shift these two references, keeping *prev* one step before *cur*. We continue until *cur* reaches the node which we need to delete. There are three exceptional cases, we need to take care of:

  1. list is empty
  2. delete the head node
  3. node is not in the list



```
public void remove(AnyType key)
{
   if(head == null) throw new RuntimeException("cannot delete");

   if( head.data.equals(key) )
   {
      head = head.next;
      return;
   }

   Node<AnyType> cur  = head;
   Node<AnyType> prev = null;

   while(cur != null && !cur.data.equals(key) )
   {
      prev = cur;
      cur = cur.next;
   }

   if(cur == null) throw new RuntimeException("cannot delete");

   //delete cur node
   prev.next = cur.next;
}
```

- Step 1: **Single Linked List Implementation**
  1. Create a SingleLinkedList.java file
  2. Refer to previous codes to define a “Node” class in it.
  3. Refer to previous codes to implement addFirst, etc method;
- Step 2: **Test Cases**
  1. Build a linked list as following and *display()* it.
     $A - - > B - - > C - - > D$
  2. Remove “B”;
  3. Insert “Tom” after “C”.



```
We construct a brand new linked list:
-->A-->B-->C
After B is removed we get a new one:
-->A-->C
Add D after C we get:
-->A-->C-->D
Insert Tom after C we get:
-->A-->C-->Tom-->D
```

Thank you!

AUTHOR:	Longsheng Zhou
ADDRESS:	ICT 609e
	Department of Computer Science
	University of Calgary
EMAIL:	lozhou@ucalgary.ca