Linear Lists and Linked Lists

Recall:

- To store a list of data, we can implement a list using an array

- However, for many applications, arrays are inconvenient.
  - E.g. it is difficult to insert and delete elements

Abstract Data Type (ADT)

- ADT is a collection of data together with a set of operations that can be performed on a data.

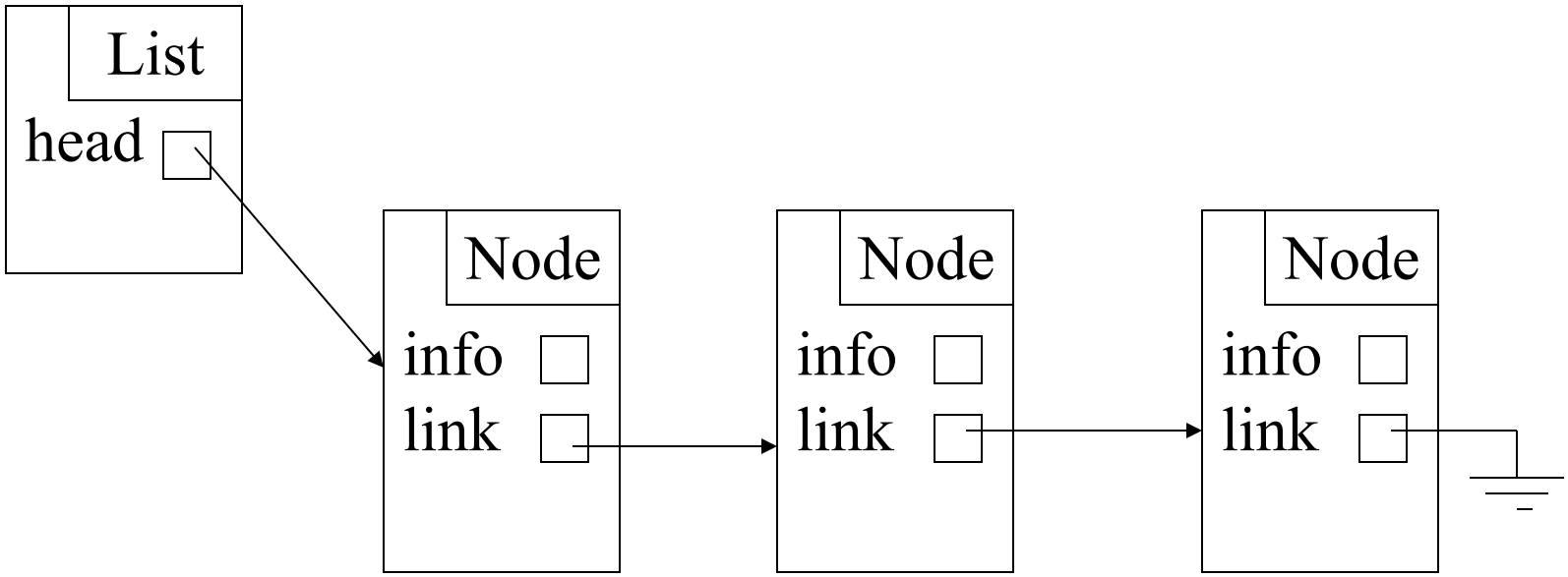
Linear List

- A linear list ADT is a *sequence* of nodes along with a set of operations for these nodes.
  - The "node"contains the data for each element in the ADT
- The sequence of a linear list can be written as:

  $x_1, x_2, x_3, \ldots\ldots, x_n$

  where x_1 is the first node, x_2 is the second node, and x_n is the last node of the linked list

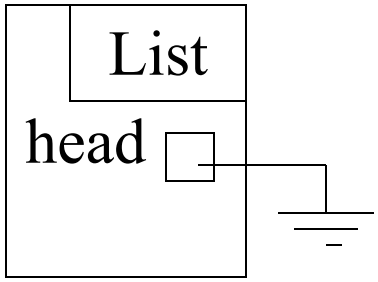
An illustration of a List



Defining your list

```
class List
{
    private student head;
    ……
```

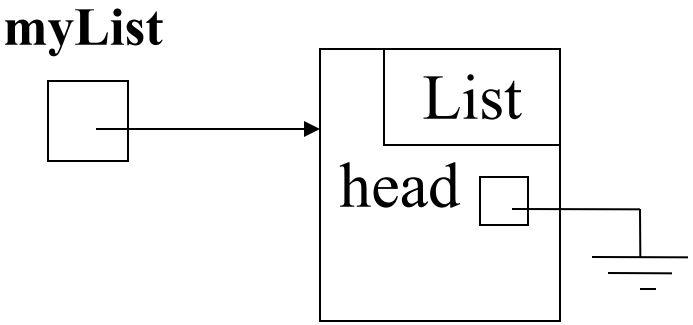
- Defines the head (and only the head) of a linked list as illustrated below. NOTE: This only defines a list, it does not create a list.



Creating your list

- To create an actual empty list based on the prior class definition,
  - In your main program, you can write

  List myList = new List();
  - Where myList is the name of the List.
  - The result of that command is…..



Adding a Node

Defining a Node

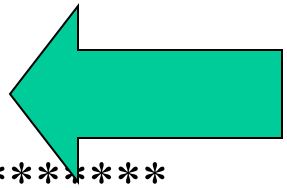
- Defining the structure of each node in a linked list is very similar to defining an object.

- The difference is in addition to the class fields, we need to add an additional field that will link to the next node.

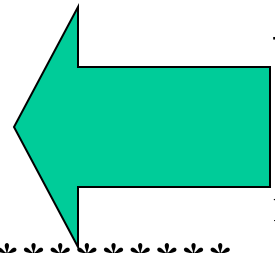
Defining a Node

- In your class List,

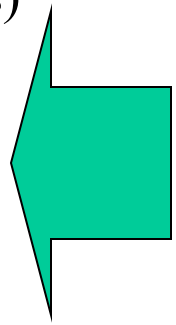
```
class List
{
    private student head;
    //**********************
    class student
    {
        int mark
        student link;
        //******************
        student (int i, student s)
        {
            mark = i;
            link = s;
        }

    }
}
```



Defines the head of a linked list



- Fields can only be accessed through List
-Defines any additional nodes
- This one contains a field called mark and a field called link



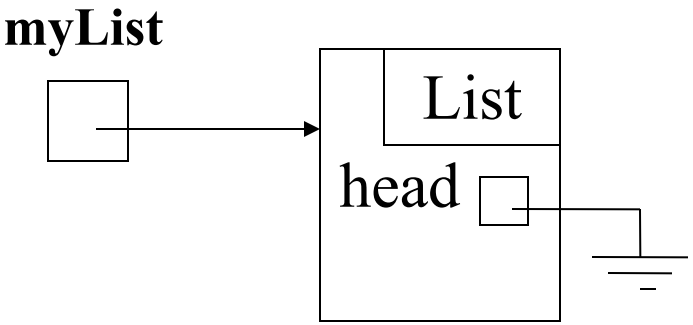
Constructor for a node

Inserting a Node to our list

- Recall:

  List myList = new List();

  // will create an empty list



Inserting a Node to our list

- The following instance method in our class **List** will insert a node at the end of the Linked List

```
public void insertFirst (int item)
{
        head = new student (item, null);
}
```

- The following call will create a ‘student’ at the end of our Linked List (which so far is empty)

```
myList.insertFirst(75);
```

```
class List{
    private student head;

    class student{
        int mark //class fields
        student link;

        student (int i, student s) {
            //constructor
            mark = i;
            link = s;
        }

    }

    public void insertFirst (int item) {
        head = new student (item, null);
    }
}
```

insertFirst is an instance method for *List* NOT *student*

Inserting a second Node to our list

- The following instance method in our class List will insert a second node at the end of the Linked List

```
public void insertSecond (int item)
{
        head.link = new student (item, null);
        // .link refers to the link field of the first node
}
```

- The following call will create a ‘student’ at the end of the first node

```
myList.insertSecond(60);
```

Inserting a third Node to our list

- The following instance method in our class List will insert a third node at the end of the Linked List

```
public void insertThird (int item)
{
        head.link.link = new student (item, null);
        // .link.link refers to the link field of the second node
        // hence we have two '.links'!
}
```

- The following call will create a 'student' at the end of the second node

```
myList.insertThird(40);
```