Linked Lists

- Basic linked list
- Node Class
- List of Node Class
- Interfaces

Node Class

- Each node contains two fields:
  - data (the info, could be an Object)
  - link (one or more) to another node

```
public class Node {
        private E data;
        private Node<E> next;
}
```

- The class has other methods like getData(), setData(), getNext(), setNext(), and a constructor

Node Class

- Later on, we will make this class more complex by making the data field a real object!

```
public class Node {
        private Flight data;
        private Node<Flight> next;
}
```

Where Flight is a class representing flight information: airline, departure and arrival times/dates, airport, etc.

Node Class

```
public class Node {
        private E data;
        private Node<E> next;
        public Node(E data, Node<E> next){
                this.data = data;
                this.next = next;
        }
        public Node(E d){
                data = d;
                next = null;
        }
}
```

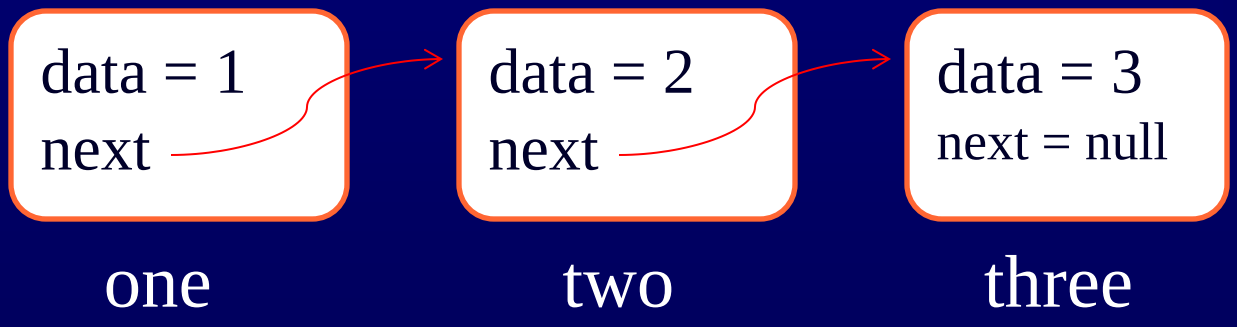
Node Class

```
public class Node {
        private int data;
        private Node next;
        public Node(int data, Node next){
                this.data = data;
                this.next = next;
        }
        public Node(int n){
                data = n;
                next = null;
        }
}
```

Creating a List

- To create a list, we need to "link" one node to another.

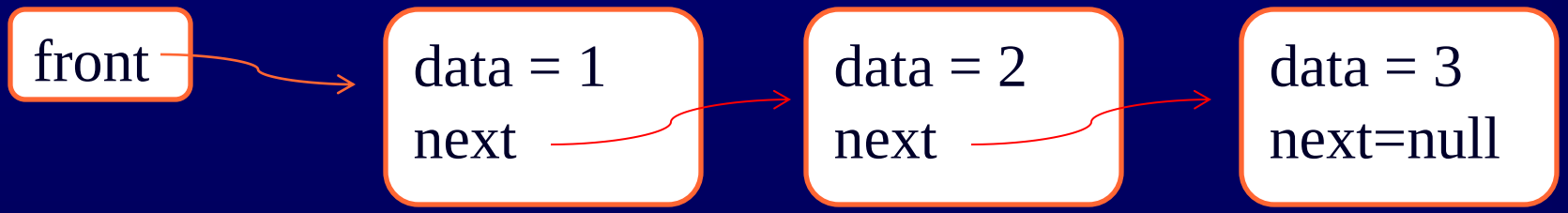
```
Node one = new Node(1);
Node two = new Node(2);
Node three = new Node(3);
one.next = two;
two.next = three;
```



Creating the list

- all we need to know is where the first node is.
  - Using "the chain" we can get to any other node
  - But we can only go in one direction!

- We use a node (of the same class) to "mark" the beginning of the list. Usually, it's called "head" or "front".



Using temp within a list …

Assume

Node temp = some node in the linked list;

This code WILL modify our list:

front = … ;

temp.next = … ;

temp.data = … ;

This code WILL NOT modify our list:

temp = … ;

Using the value ‘null’

- Use it to indicated that a reference is not being used, as in

```
front = null;          //the empty list
front.next= null       //list with 1 node
```

- different than setting an String to null (empty string) because I still can compare it to another String
- different than setting an array to null (empty) because I still can ask for its length
- different than setting an ArrayList to null (empty) because I still can add elements to it

List Interface in Java

Java's List<E> interface:

```
int size()
boolean add(E obj)
void add(int index, E obj)
E get(int index)
E set(int index, E obj)
E remove(int index)
```

A List<E> embodies the idea that we have a sequence of elements that can be accessed by index.

List Class implementation

Java implements the List Class with two implementations:

ArrayList<E>
LinkedList<E>

Behavior is the same, implementation (run times) is different.

ArrayList : array :: LinkedList : linked list nodes

LinkedList Class run times

LinkedList's size() runs in O(1) time. How?

LinkedList's add(E obj) runs in O(1) time. How?

LinkedList's remove(int index) runs in O(1) time when deleting the last node. How?

- what about deleting the first node?
- what about deleting a node in the middle?

Node Class (circular list)

- Each node contains three fields:
  - data (the info, an Object)
  - links to the nodes before and after

```
public class Node {
        private E data;
        private Node<E> next;
        private Node<E> prev;
}
```

LinkedList Class

- Is it single or double?
- Is it circular?
- Does it have a *front* reference?
- Does it have a *tail* reference?
- Does it have a counter?

- Let’s draw a picture …

LinkedList Class run times

What's the running time for

- get (int index) ?
- set (int index, E obj) ?
- add(int index, E obj) ?

Double Linked Lists

- Each node contains three fields:
  - data (the info)
  - two references: next & previous

```
public class Node {
        private int data;
        private Node next;
        private Node previous;
}
```

Circular Linked Lists

- The last node refers to the first node
  - Where is the beginning?
  - Node still is the same
- Could use a “single” or a “double” linked list

Homework

- Homework # 2 (flights) due tonight
- Quiz #2 (ArrayList) tomorrow in recitation
- Homework # 3 (linked lists) due Tuesday 9/22

Readings

- Single Linked Lists
- Double Linked Lists

- Java API for List interface
- Java API for LinkedList Class