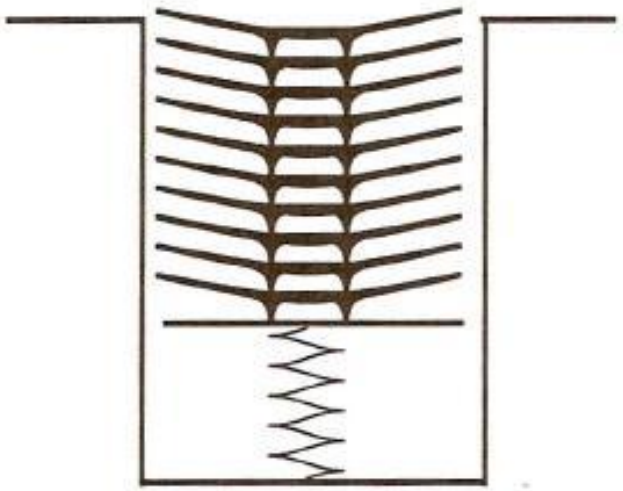
Lesson 3 – Stacks and Queues

- For many applications, the insertion and deletion of items are only required at one ends of a list.

Stacks

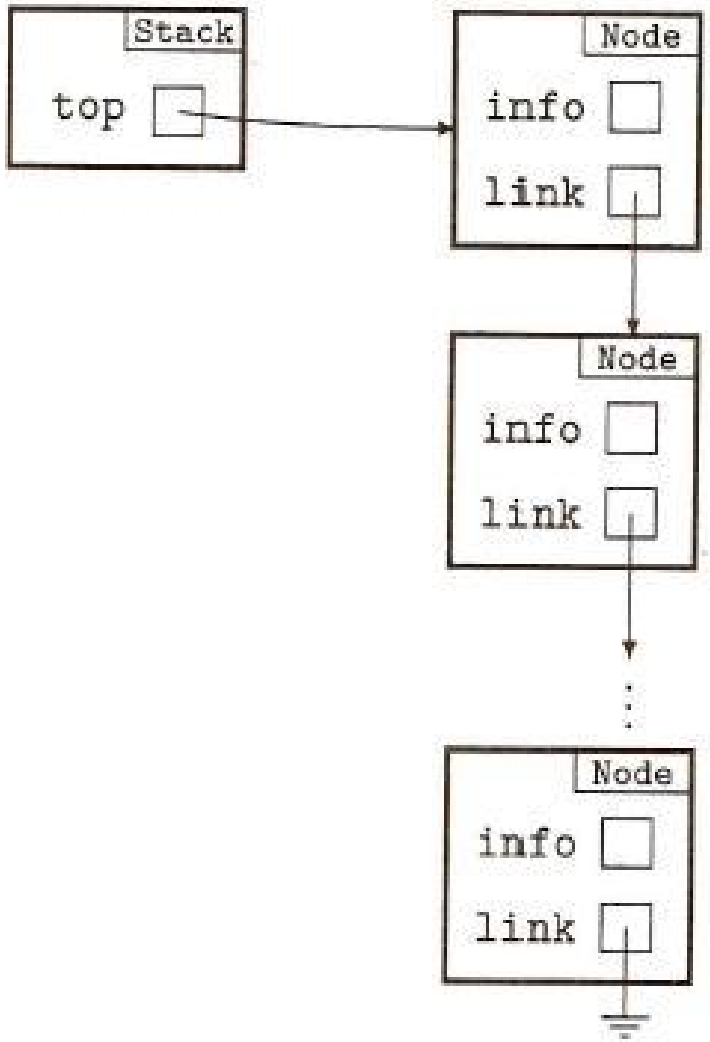
- A linear list where insertions, deletions, and checks take place at the top of a list.
  - The other end is called the bottom of the stack
  - LIFO – Last in First Out



5 Functions in a Stack

1. Create – Create a new empty stack
2. Push – push (or insert) item onto the top of the stack
3. Pop – pop (or remove) item from the top of the stack
4. isEmpty – checks if the stack is empty – returns true if so
5. peek – return the value of the item at the top.

Implementation



Example 1 - Pushing an item

```
//this method inserts an item to a stack

public void push (int item)
{
    top = new Node(item,top);
}
```

Example 2

// this method returns the value currently at the top of the stack without altering the stack.

```
public int peek ()
{
    if (top == null)
        throw new RuntimeException("peek: emptystack");
    else
        return top.info;
}
```

- A stack class in the package java.util will contain methods for the operations of a stack.

Queue

A standard Line up

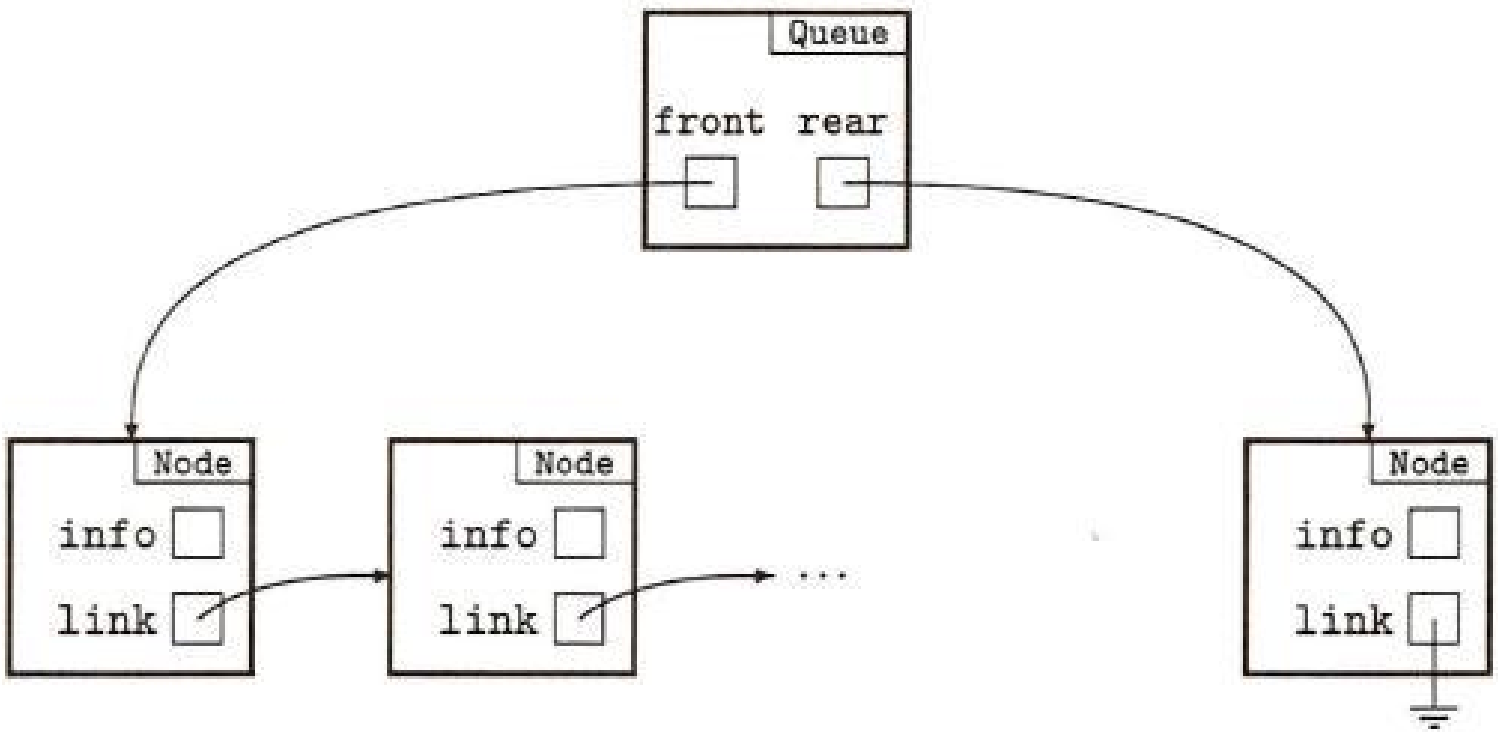
- Operates like a queue at a checkout line at a store.
  - Insertions take place at one end of the queue (the rear of the queue)
  - Deletions take place at the other end. (the front of the queue)
- FIFO – First in First Out

Functions in a Queue

1. Create – create a new empty queue
2. enqueue – add a new item to the rear of the list
3. dequeue – delete and remove the item at the front of the list
4. isEmpty – return true if the queue is empty
5. peek – return the value of the item at the front

Implementation

- Because insertions and deletions occur at opposite ends of the queue, have separate references to the front and rear of queue



The Queue class

- Create a Queue class by setting up fields for the references to the two ends of the lists.

code

```
class Queue
{
    private Node front;
    private Node rear;

    class Node
    {
        int info;
        Node link;

        Node (int i, Node n)
        {
            info = i;
            link = n;
        }
    }
}
```

Adding a node to a queue

- 2 cases
  1. When the queue is empty, both *front* and *rear* must refer to the new node
  2. New node must be inserted after the node referred to by *rear*, and *rear* must now refer to the new node

Example 3 - Code

```
// method inserts a new node into a lost of the Queue class
public void enqueue (int item)
{
    Node temp = new Node(item,null);
    if (rear == null)
        // queue was empty
        front = rear = temp;
    else
        // add node at rear of queue
        rear = rear.link = temp;
}
```