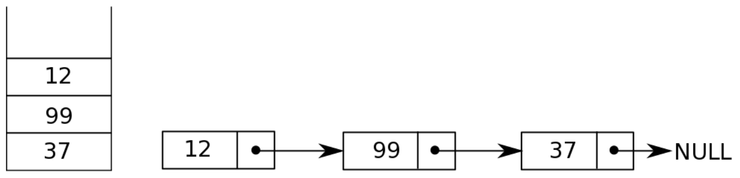
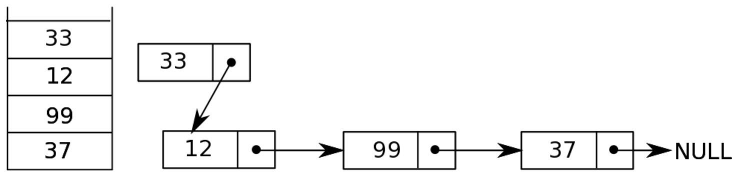
Node-based Stack

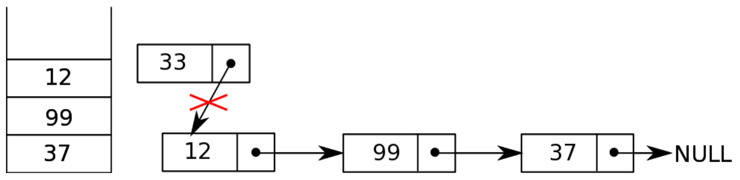
- Another way to implement a growing stack.
- Consider every element in a stack is a node. Each node has two fields: data and the pointer/reference points to the node before it. If there is no node before it, the pointer is set to `NULL`.
- The principle benefit of this node-based stack is that the stack elements can easily be inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory.
  - But it uses more memory than the array-based stack due to their pointers.



Push



Pop



- Implementation
  - We need to implement a generic Node class (private inner helper class).
    - A nested class has access to all the private members of its enclosing class—both fields and methods. Therefore, a public or protected nested class inherited by a subclass has indirect access to all of the private members of the superclass.
  - The stack has a instance variable called top that points to the element on top of the stack, and an int variable keeping the size of the stack.
  - The constructor creates an empty stack by initializing `top` to `null` and `size` to `0`. So naturally we can test for an empty stack.

```
/**
 * NodeBasedStack.java: Implement a node based stack
 * Ying Li
 * 02/12/2018
 */

import java.util.EmptyStackException;

public class NodeBasedStack<T> implements StackInterface<T> {
     // a private Node class
```

```
    private class Node {
        T data;
        Node next;

        public Node (T val, Node n) {
            data = val;
            next = n;
        }

        public void setData (T val) {
            data = val;
        }

        public T getData () {
            return data;
        }

        public Node getNext () {
            return next;
        }

        public void setNext (Node n) {
            next = n;
        }
    }
    // end of private Node class

    // field of NodeBasedStack class
    Node top;
    int size;

    // constructor of the NodeBasedStack class
    public NodeBasedStack () {
        top = null;
        size = 0;
    }

    // check if the stack is empty
    public boolean isEmpty () {
        return top == null;
    }

    // get the number of elements in the stack
    public int size () {
        return size;
    }

    // add a new element to the top of the stack
    public void push (T val) {
        Node node = new Node(val, top);
        top = node;
        size++;
    }

    // return the top element and remove it from the stack
    public T pop () {
        T data = null;
        if (isEmpty()) throw new EmptyStackException();
        else {
            data = top.getData(); // get the data stored in the top element
            Node tmp = top; // set a tmp pointer points to the top element
            top = top.getNext(); // update the top pointer
            tmp.setNext(null); // update the next pointer of the original top
element for GC
```

```
                size--;
            }
            return data;
        }

        // return the value of the top element
        public T peek () {
            T data = null;
            if (isEmpty()) throw new EmptyStackException();
            else data = top.getData(); // get the data stored in the top element
            return data;
        }

        public static void main (String[] args) {
            NodeBasedStack<Integer> stack = new NodeBasedStack<Integer>();
            stack.push(37);
            stack.push(99);
            stack.push(12);

            System.out.println("After push, size of the stack: " + stack.size());
            System.out.println("Top is: " + stack.peek());

            System.out.println("*** Testing pop ***");
            while (!stack.isEmpty()){
                System.out.println(stack.pop());
            }

            System.out.println("After pop, size of the stack: " + stack.size());
        }
    }
```

- Please note:
  - Node class should be generic. We must place <T> carefully. A way to check whether all necessary <T> have been placed in your code is by using `javac -Xline`.
  - It is inappropriate to use for loop based on the size of the stack in the main method, as the size of the stack keeps changes (due to the pop). Using while loop is more appropriate.

- Java API also provides a Stack class [https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html]. It has methods `empty`, `peek`, `push`, pop, and `search`. You can try to implement the search method for each of the above three approaches. Java Stack is also a Generic Object type. So, we need to specify the type if we want to create a Stack object.