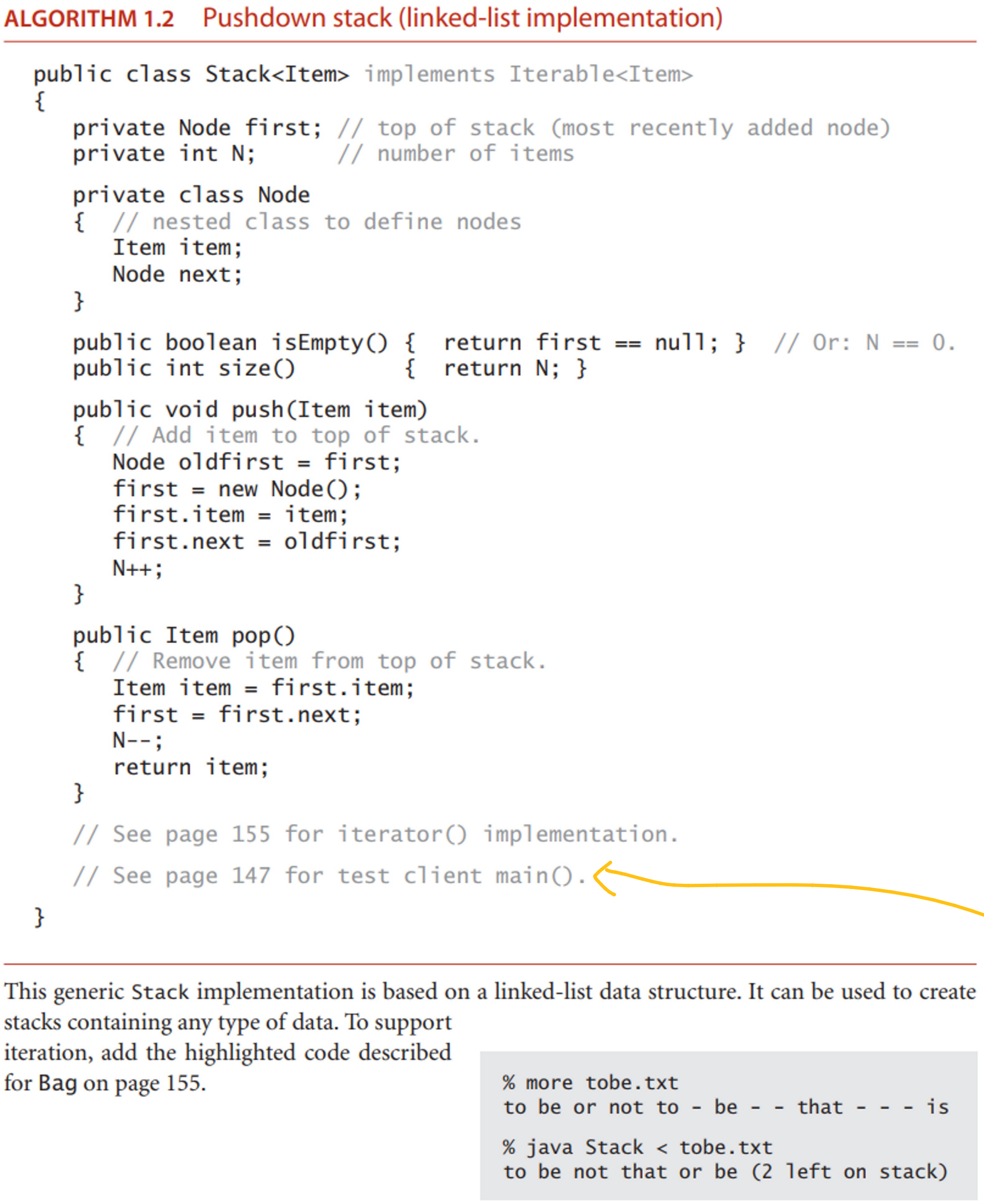
Stack in a linked list

Monday, September 12, 2022 12:56 PM

Instead of making a stack an array, we can maintain the stack as a linked list, where insertions and deletions of items becomes more easier and efficentc compare to an array which everytime adding and deleting items requires to create a new array to copy and paste the previous elements.

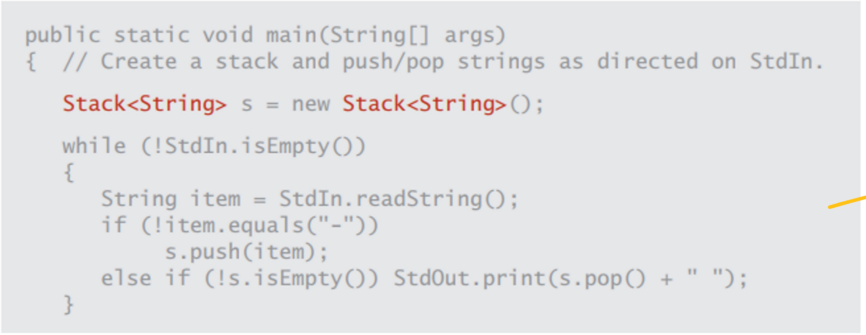


ALGORITHM 1.2 Pushdown stack (linked-list implementation)

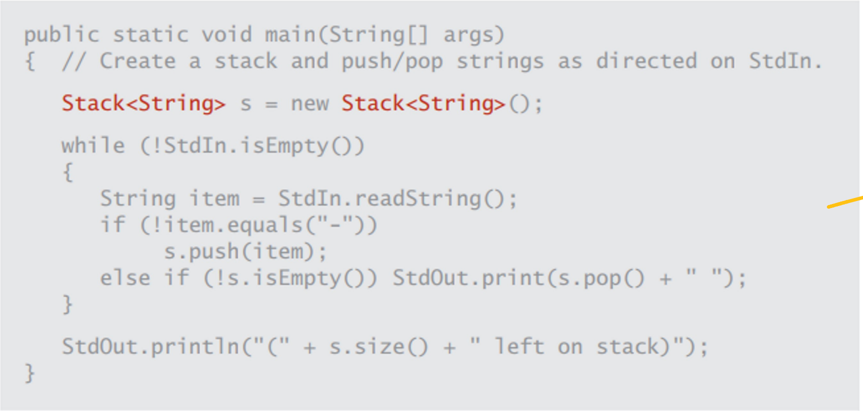
```
public class Stack<Item> implements Iterable<Item>
{
   private Node first; // top of stack (most recently added node)
   private int N;      // number of items

   private class Node
   {  // nested class to define nodes
      Item item;
      Node next;
   }

   public boolean isEmpty() {  return first == null; }  // Or: N == 0.
   public int size()        {  return N; }

   public void push(Item item)
   {  // Add item to top of stack.
      Node oldfirst = first;
      first = new Node();
      first.item = item;
      first.next = oldfirst;
      N++;
   }

   public Item pop()
   {  // Remove item from top of stack.
      Item item = first.item;
      first = first.next;
      N--;
      return item;
   }

   // See page 155 for iterator() implementation.

   // See page 147 for test client main().
}
```

This generic Stack implementation is based on a linked-list data structure. It can be used to create stacks containing any type of data. To support iteration, add the highlighted code described for Bag on page 155.

```
% more tobe.txt
to be or not to - be - - that - - - is

% java Stack < tobe.txt
to be not that or be (2 left on stack)
```



```
public static void main(String[] args)
{  // Create a stack and push/pop strings as directed on StdIn.
   Stack<String> s = new Stack<String>();

   while (!StdIn.isEmpty())
   {
      String item = StdIn.readString();
      if (!item.equals("-"))
           s.push(item);
      else if (!s.isEmpty()) StdOut.print(s.pop() + " ");
   }
```



```
public static void main(String[] args)
{  // Create a stack and push/pop strings as directed on StdIn.
   Stack<String> s = new Stack<String>();
   while (!StdIn.isEmpty())
   {
      String item = StdIn.readString();
      if (!item.equals("-"))
           s.push(item);
      else if (!s.isEmpty()) StdOut.print(s.pop() + " ");
   }
   StdOut.println("(" + s.size() + " left on stack)");
}
```

Test client for Stack

Above is the optimum design of a stack class:

- It can be used for any type of data
- The space required is always proportional to the size of the collection.
- The time per operation is always independent of the size of the collection.