Public and Private

```
public class Date {
 private int day;
 private int month;
     private void setMonth(int m)
     month = m;
    }
public Date(int month, int day) {
Implementation includes error-checking
}
```

```
public class TamperMonkey {
public void tamper() {
Date d = new Date(9, 25);
d.day = 75;     // Will it work?
d.setMonth(20);  // Will it work
}
```

Will TamperMonkey compile?

Abstract Data Type

The ***Interface*** of a class =

A set of public methods +

Descriptions of the methods' behaviors

(but not how they are implemented).

An **Abstract Data Type (ADT)** is a well-defined Interface without any details about its implementation. Treat this as a user-defined data-type. An ADT as a mathematical model of the data objects that make up a data type as well as functions that operate on these objects.

Some examples are:

- List
- Stack
- Queue
- Tree
- Heap

The List ADT

Here is a sample list:

`Bread cheese, tea, coffee, milk, honey, pizza,`

Java defines a general interface `java.util.List` that includes the following index-based methods (since that provides more general support for addition or deletion of items) and many more

`Size ()`	`return the SIZE`
`isEmpty()`	`returns TRUE or FALSE`
`get(i)`	`returns element with index i`
	Error occurs when index is outside the range
`set(i, e)`	`updates element i to e`
	Error occurs when index is outside the range
`add (i, e)`	`inserts item e after element with index i`
	Error occurs when index is outside the range
`remove (i)`	`deletes the` i^{th} `element`
	Error occurs when index is outside the range

Stack ADT

What is a stack?

What are the invariants?

From abstract to concrete

One way to implement the list ADT is to use an array. ArrayList creates the illusion of an unbounded array, by repeatedly copying fixed size arrays into a larger space when new elements are inserted.

Public class ArrayList <E> **implements** list <E>

There can be other implementations of the list ADT.

Each ADT should have one or more invariants that are true, regardless of the implementation.

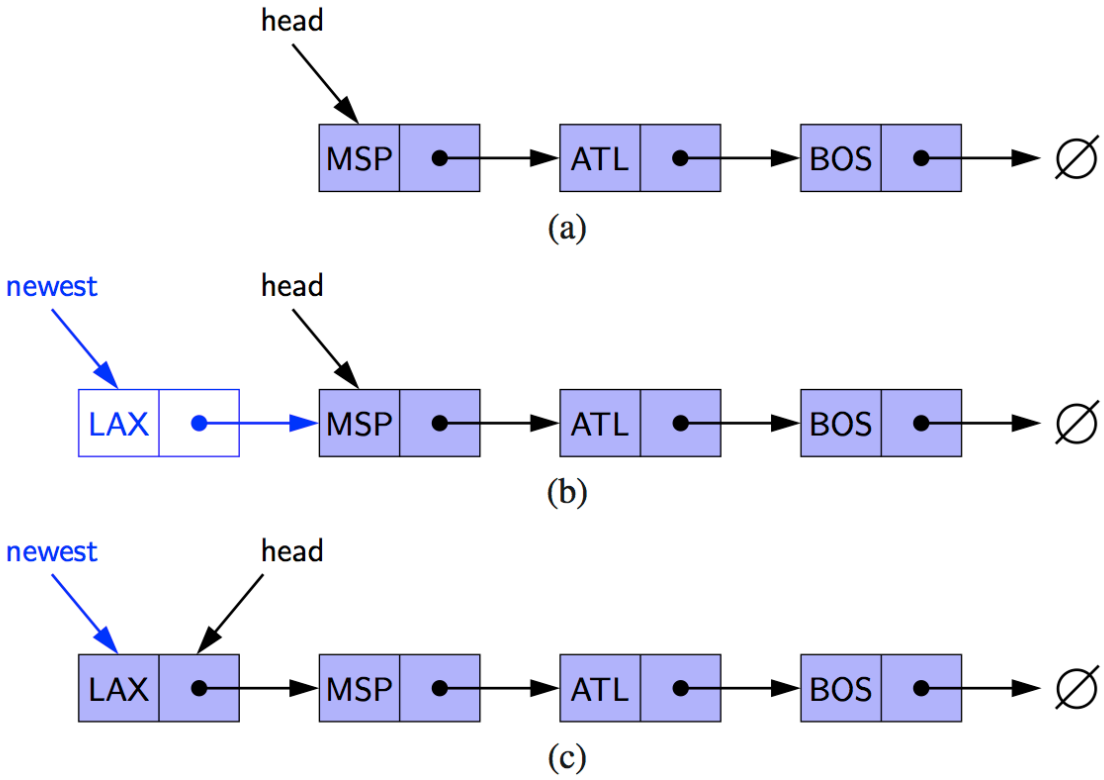
What is the invariant of a list ADT?

"There is always a tail " (so no circular structure)

The Singly Linked List

```
 1  public class SinglyLinkedList<E> {
...   (nested Node class goes here)
14    // instance variables of the SinglyLinkedList
15    private Node<E> head = null;          // head node of the list (or null if empty)
16    private Node<E> tail = null;          // last node of the list (or null if empty)
17    private int size = 0;                 // number of nodes in the list
18    public SinglyLinkedList( ) { }        // constructs an initially empty list
19    // access methods
20    public int size( ) { return size; }
21    public boolean isEmpty( ) { return size == 0; }
22    public E first( ) {                   // returns (but does not remove) the first element
23      if (isEmpty( )) return null;
24      return head.getElement( );
25    }
26    public E last( ) {                    // returns (but does not remove) the last element
27      if (isEmpty( )) return null;
28      return tail.getElement( );
29    }
```

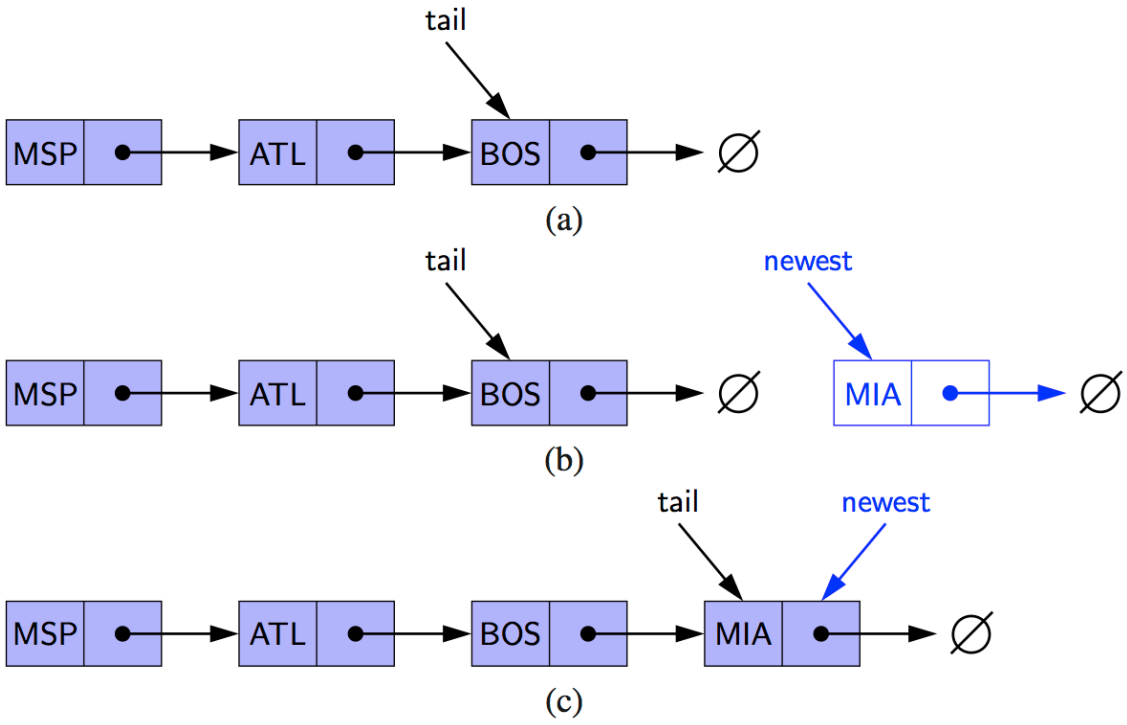
Inserting at the Head



Removing the Head

Removing the tail : why is it slow?

Inserting at the tail

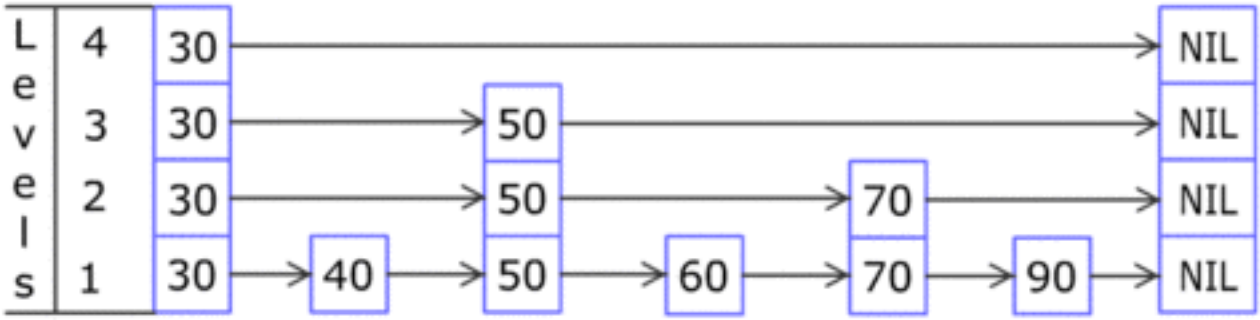


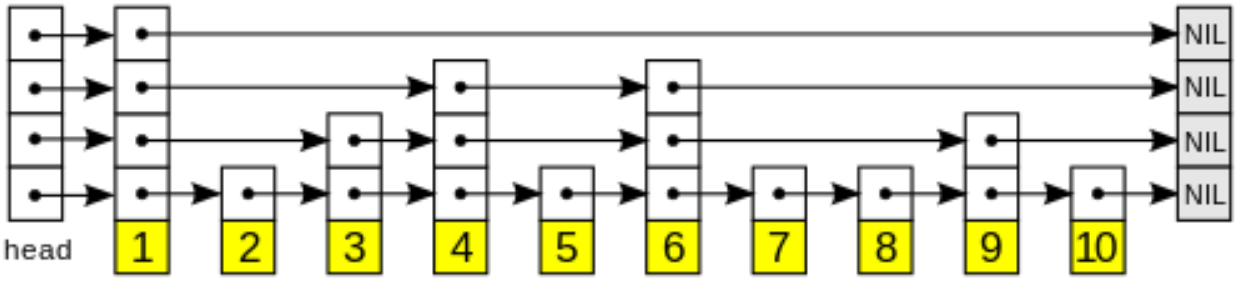
Doubly Linked List

Circular Linked Lists

Skip List

Helps manage a list efficiently





Figures taken from Wikipedia