SLL: Constructing a Chain of Nodes

```
public class Node {
   private String element;
   private Node next;
   public Node(String e, Node n) { element = e; next = n; }
   public String getElement() { return element; }
   public void setElement(String e) { element = e; }
   public Node getNext() { return next; }
   public void setNext(Node n) { next = n; }
}
```

Approach 1

```
Node tom = new Node("Tom", null);
Node mark = new Node("Mark", tom);
Node alan = new Node("Alan", mark);
```

SLL: Constructing a Chain of Nodes

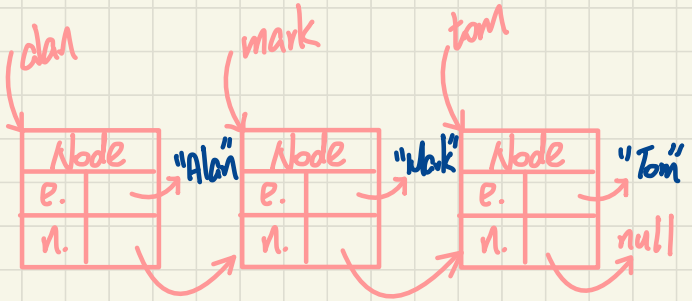
```
public class Node {
   private String element;
   private Node next;
   public Node(String e, Node n) { element = e; next = n; }
   public String getElement() { return element; }
   public void setElement(String e) { element = e; }
   public Node getNext() { return next; }
   public void setNext(Node n) { next = n; }
}
```

Approach 2

```
Node alan = new Node("Alan", null);
Node mark = new Node("Mark", null);
Node tom = new Node("Tom", null);
alan.setNext(mark);
mark.setNext(tom);
```

SLL: Setting a List's Head to a Chain of Nodes

```
public class SinglyLinkedList {
   private Node head = null;
   public void setHead(Node n) { head = n; }
   public int getSize() { ... }
   public Node getTail() { ... }
   public void addFirst(String e) { ... }
   public Node getNodeAt(int i) { ... }
   public void addAt(int i, String e) { ... }
   public void removeLast() { ... }
}
```

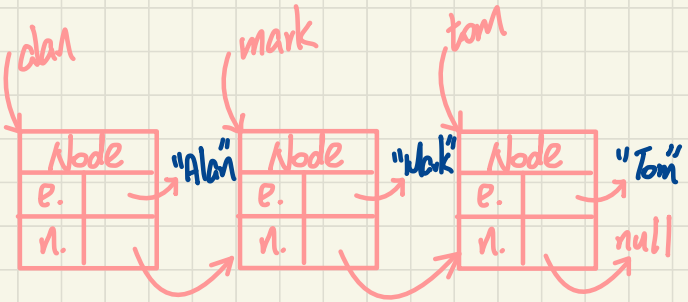


Approach 1

```
Node tom = new Node("Tom", null);
Node mark = new Node("Mark", tom);
Node alan = new Node("Alan", mark);
SinglyLinkedList list = new SinglyLinkedList();
list.setHead(alan);
```

SLL: Setting a List's Head to a Chain of Nodes

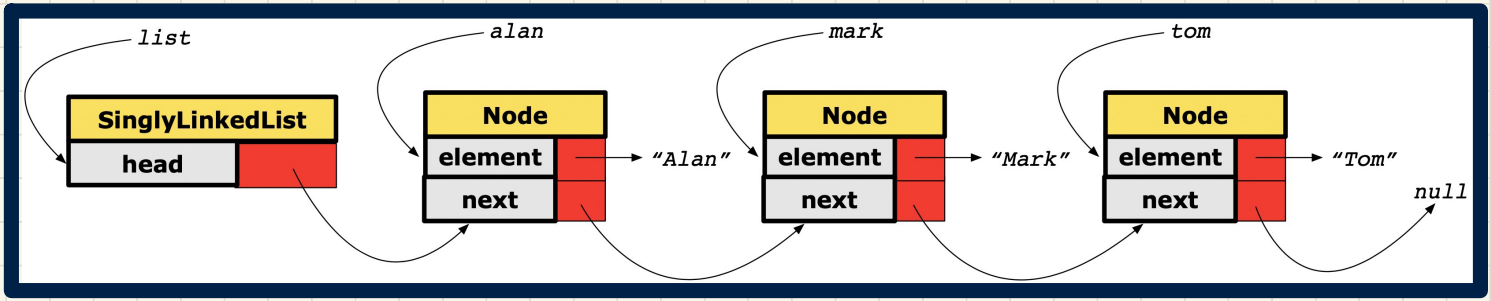
```
public class SinglyLinkedList {
   private Node head = null;
   public void setHead(Node n) { head = n; }
   public int getSize() { ... }
   public Node getTail() { ... }
   public void addFirst(String e) { ... }
   public Node getNodeAt(int i) { ... }
   public void addAt(int i, String e) { ... }
   public void removeLast() { ... }
}
```



Approach 2

```
Node alan = new Node("Alan", null);
Node mark = new Node("Mark", null);
Node tom = new Node("Tom", null);
alan.setNext(mark);
mark.setNext(tom);
SinglyLinkedList list = new SinglyLinkedList();
list.setHead(alan);
```

SLL Operation: Counting the Number of Nodes

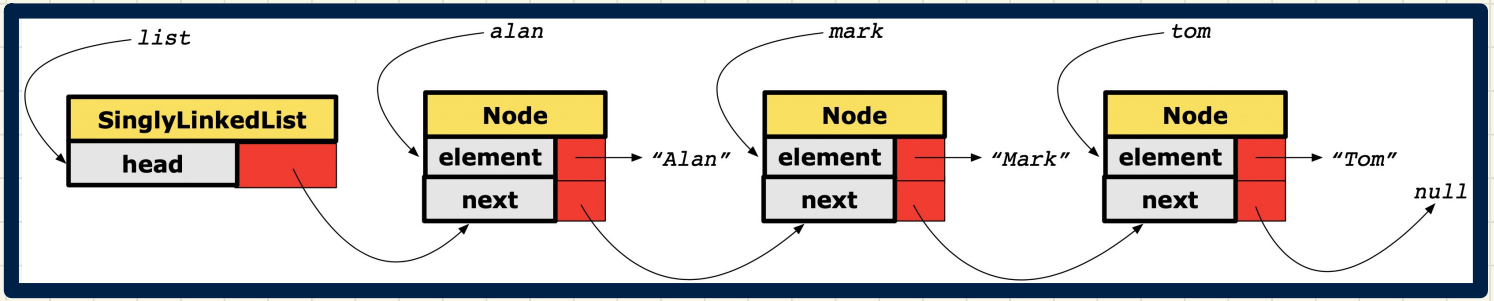



```
1  int getSize() {
2     int size = 0;
3     Node current = head;
4     while (current != null) {
5        current = current.getNext();
6        size ++;
7     }
8     return size;
9  }
```

Trace: list.getSize()

current	current != null	End of Iteration	size

SLL Operation: Finding the Tail of the List

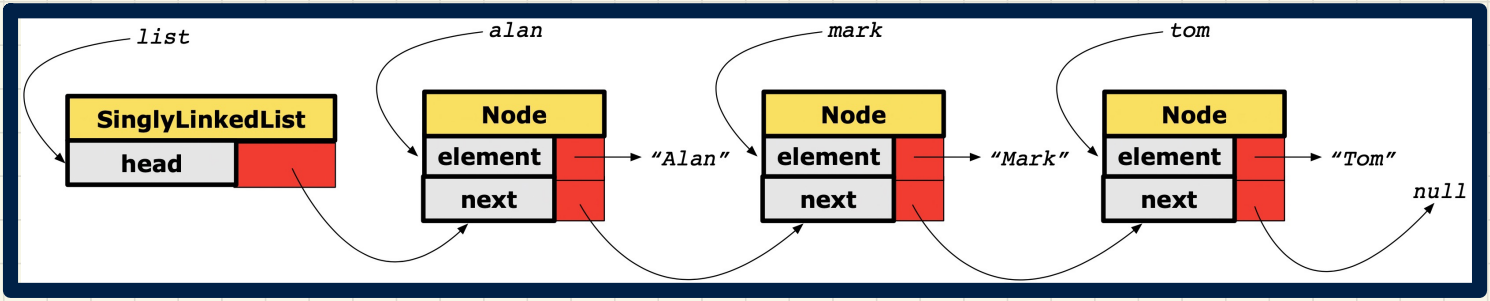



```
1  Node getTail() {
2     Node current = head;
3     Node tail = null;
4     while (current != null) {
5        tail = current;
6        current = current.getNext();
7     }
8     return tail;
9  }
```

Trace: list.getTail()

current	current != null	End of Iteration	tail

SLL: Trading Space for Time



SLL Operation: Inserting to the Front of the List

```java
@Test
public void testSLL_02() {
    SinglyLinkedList list = new SinglyLinkedList();
    assertTrue(list.getSize() == 0);
    assertTrue(list.getFirst() == null);

    list.addFirst("Tom");
    list.addFirst("Mark");
    list.addFirst("Alan");
    assertTrue(list.getSize() == 3);
    assertEquals("Alan", list.getFirst().getElement());
    assertEquals("Mark", list.getFirst().getNext().getElement());
    assertEquals("Tom", list.getFirst().getNext().getNext().getElement());
}
```

```java
1  void addFirst (String e) {
2    head = new Node(e, head);
3    if (size == 0) {
4      tail = head;
5    }
6    size ++;
7  }
```

Q. Does **tail** or **size** need to be updated?

Non-Generic Classes: Node vs. SinglyLinkedList

```
public class Node {
   private String element;
   private Node next;
   public Node(String e, Node n) { element = e; next = n; }
   public String getElement() { return element; }
   public void setElement(String e) { element = e; }
   public Node getNext() { return next; }
   public void setNext(Node n) { next = n; }
}
```

```
public class SinglyLinkedList {
   private Node head = null;
   public void setHead(Node n) { head = n; }
   public int getSize() { ... }
   public Node getTail() { ... }
   public void addFirst(String e) { ... }
   public Node getNodeAt(int i) { ... }
   public void addAt(int i, String e) { ... }
   public void removeLast() { ... }
}
```

```
Node n1 = new Node("Alan", null);
Node n2 = new Node("Mark", n1);
Node n3 = new Node(23, null);
SLL list = new SLL();
list.setHead(n2);
list.addAt(0, "Tom");
list.addAt(1, 23);
Node n4 = list.getNodeAt(1);
String e = n4.getElement();
```

Generic Classes: Node and SinglyLinkedList

```
public class Node< E > {
  private E element;
  private Node< E > next;
  public Node( E e, Node< E > n) { element = e; next = n; }
  public E getElement() { return element; }
  public void setElement( E e) { element = e; }
  public Node< E > getNext() { return next; }
  public void setNext(Node< E > n) { next = n; }
}
```

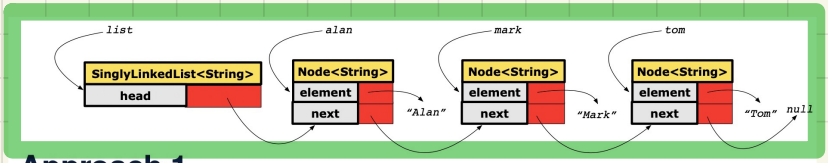
```
public class SinglyLinkedList< E > {
  private Node< E > head;
  private Node< E > tail;
  private int size;
  public void setHead(Node< E > n) { head = n; }
  public void addFirst( E e) { ... }
  Node< E > getNodeAt (int i) { ... }
  void addAt (int i, E e) { ... }
}
```

```
Node<String> n1 = new Node<>("Alan", null);
Node<String> n2 = new Node<>("Mark", n1);
Node<Integer> n3 = new Node<>(23, null);
Node<Integer> n4 = new Node<>(46, n3);
Node<Integer> n5 = new Node<>("Tom", null);
Node<Integer> n6 = new Node<>(46, n2);

SLL<String> list1 = new SLL<>();
list1.setHead(n2);
list1.addAt(0, "Tom");
Node<String> n7 = list1.getNodeAt(1);
String e1 = n7.getElement();

SLL<Integer> list2 = new SLL<>();
list2.setHead(n4);
list2.addAt(0, 68);
Node<Integer> n8 = list2.getNodeAt(1);
Integer e2 = n8.getElement();
```

List Constructions



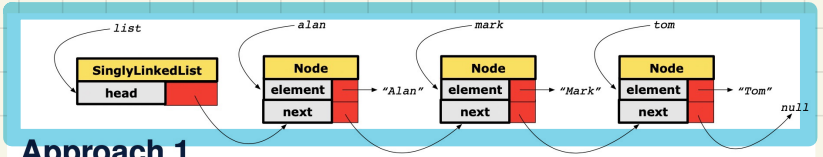
Approach 1

```
Node<String> tom = new Node<String>("Tom", null);
Node<String> mark = new Node<>("Mark", tom);
Node<String> alan = new Node<>("Alan", mark);
SinglyLinkedList<String> list = new SinglyLinkedList<>();
list.setHead(alan);
```

Approach 2

```
Node<String> alan = new Node<String>("Alan", null);
Node<String> mark = new Node<>("Mark", null);
Node<String> tom = new Node<>("Tom", null);
alan.setNext(mark);
mark.setNext(tom);
SinglyLinkedList<String> list = new SinglyLinkedList<>();
list.setHead(alan);
```

Generic List



Approach 1

```
Node tom = new Node("Tom", null);
Node mark = new Node("Mark", tom);
Node alan = new Node("Alan", mark);
SinglyLinkedList list = new SinglyLinkedList();
list.setHead(alan);
```

Approach 2

```
Node alan = new Node("Alan", null);
Node mark = new Node("Mark", null);
Node tom = new Node("Tom", null);
alan.setNext(mark);
mark.setNext(tom);
SinglyLinkedList list = new SinglyLinkedList();
list.setHead(alan);
```

Non-Generic List

List Methods

Generic List

```
void addFirst (E e) {
   head = new Node<E>(e, head);
   if (size == 0) { tail = head; }
   size ++;
}
```

```
Node<E> getNodeAt (int i) {
   if (i < 0 || i >= size) {
     throw new IllegalArgumentExcept
   else {
      int index = 0;
      Node<E> current = head;
      while (index < i) {
         index ++;
         current = current.getNext();
      }
      return current;
   }
}
```

Non-Generic List

```
void addFirst (String e) {
   head = new Node(e, head);
   if (size == 0) {
      tail = head;
   }
   size ++;
}
```

```
Node getNodeAt (int i) {
   if (i < 0 || i >= size) { /* error
   else {
      int index = 0;
      Node current = head;
      while (index < i) { /* exit when
         index ++;
         current = current.getNext();
      }
      return current;
   }
}
```