CS206

Linked List

Reference

- A reference variable holds a memory address where the referenced object is stored
  - They are usually just call “Objects”
- Reference types
  - Anything that inherits from Object (including `String`, `Integer`, `Double`, etc)
  - “primitive” types: int, float, etc are NOT reference types
    - recognizable by starting lower case
- A reference is `null` when it doesn’t refer/point to any object

References and equality

```
String s1 = new String("abc");
String s2 = new String("abc");
String s3 = s2;
String s4 = "abc";
String s5 = "abc";

System.out.println("1 equal 2 " + s1.equals(s2));
System.out.println("1==2 " + (s1==s2));
System.out.println("2==3 " + (s2==s3));
System.out.println("4==5 " + (s4==s5));
System.out.println("1==4 " + (s1==s4));
```

Nested Class

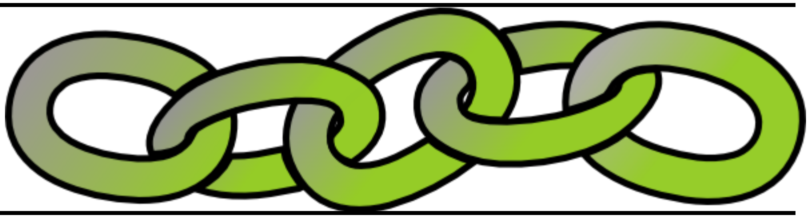
- A class defined inside the definition of another class
- Used when defining a class that is strongly affiliated with another
  - ▫ Increases encapsulation and reduces undesired name conflicts.
- Nested classes are a valuable technique when implementing data structures
  - ▫ represent a small portion of a larger data structure
  - ▫ an auxiliary class that helps navigate a primary data structure

▫ Usually private to containing class

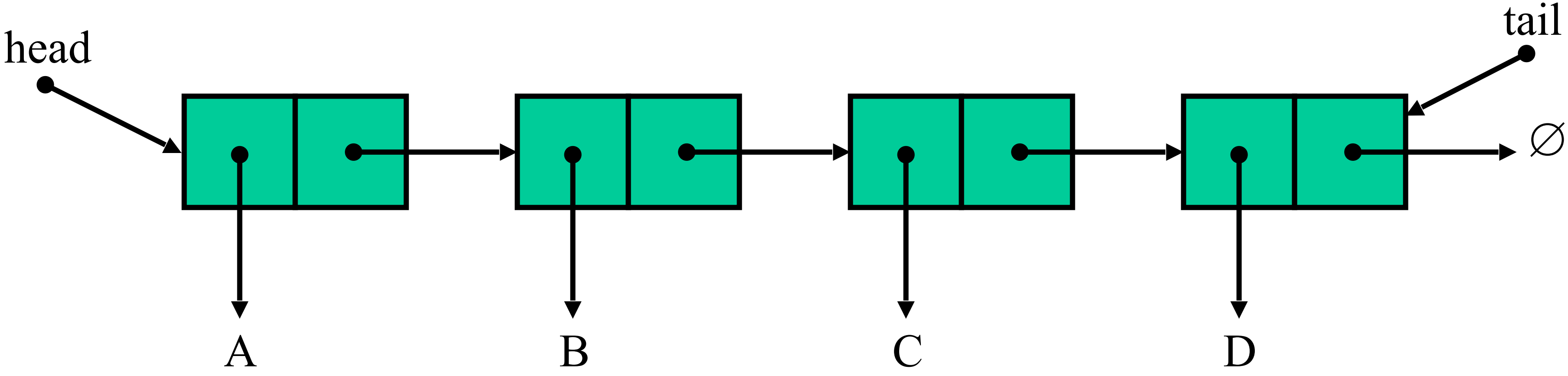
▫ Only occasion in which pubic instance variables are acceptable

  - ▫ and only when the class is strictly a data container — nothing but get & set.

Linked List

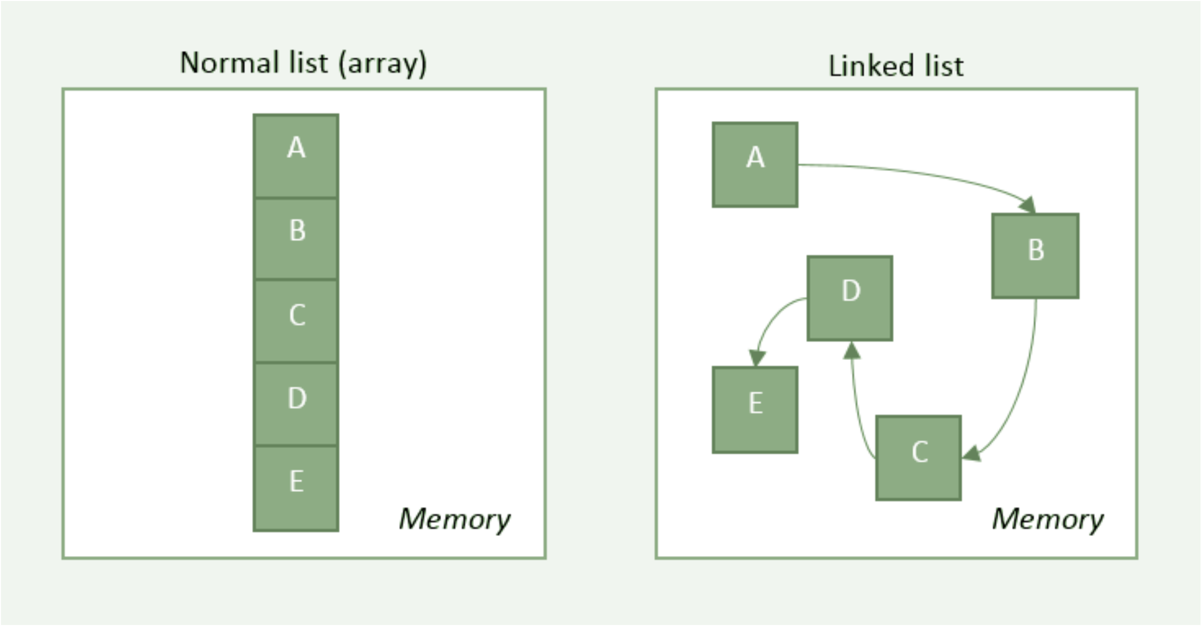


- A linked list is a lists of objects (nodes).
- The nodes form a linear sequence.
- Unbounded in length.



Linked List versus Array

- An array is a single consecutive piece of memory, a linked list is made of many disjoint pieces (the nodes).

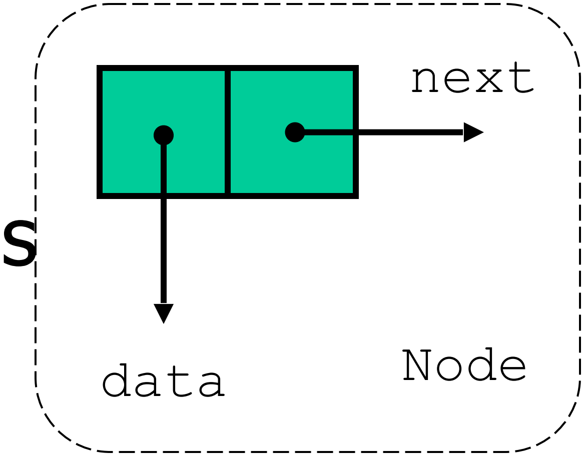


Linked List versus Array

- Array
  - quick access to any element
  - slow insertion, deletion and reordering (shifting required in general)
- Linked list
  - quick insertion, deletion and reordering of the elements
  - slow access (must traverse list)

Self-referential Structures

- A class with instance variables that reference another member of the class



```
public class Node {
   private Object data;
   private Node next;
}
```

Rabbits

You want to store data about a herd of rabbits.
Each rabbit has a breed and birthdate (stored as double) and ID.
Rabbits come and go frequently but you do not need to update rabbit data often

```
public class Rabbit {
    private String breed;
    private double birthdate;
    private String iD;

    public Rabbit(String breed, double bday, String id) {
          this.breed=breed;
          this.birthdate=bday;
         this.iD=id;
    }

    private Rabbit()  {
    }
// Other stuff
}
```

Rabbit breeds: french lop, dwarf dutch, angora, …

Node

```
private class Node {
  public Rabbit data;
  public Node next;
  public Node(Rabbit data, Node next) {
    this.data = data;
    this.next = next;
  }
}
```

A Rabbity Linked List interface

```
public interface LinkedListInterface
{
    int size();
    boolean isEmpty();
    Rabbit first();
    Rabbit last();
    void addLast(Rabbit c);
    void addFirst(Rabbit c);
    Rabbit removeFirst();
    Rabbit removeLast();
    Rabbit remove(Rabbit r);
    Rabbit find(String iD);
}
```

No mention of nodes!!

Starting Point

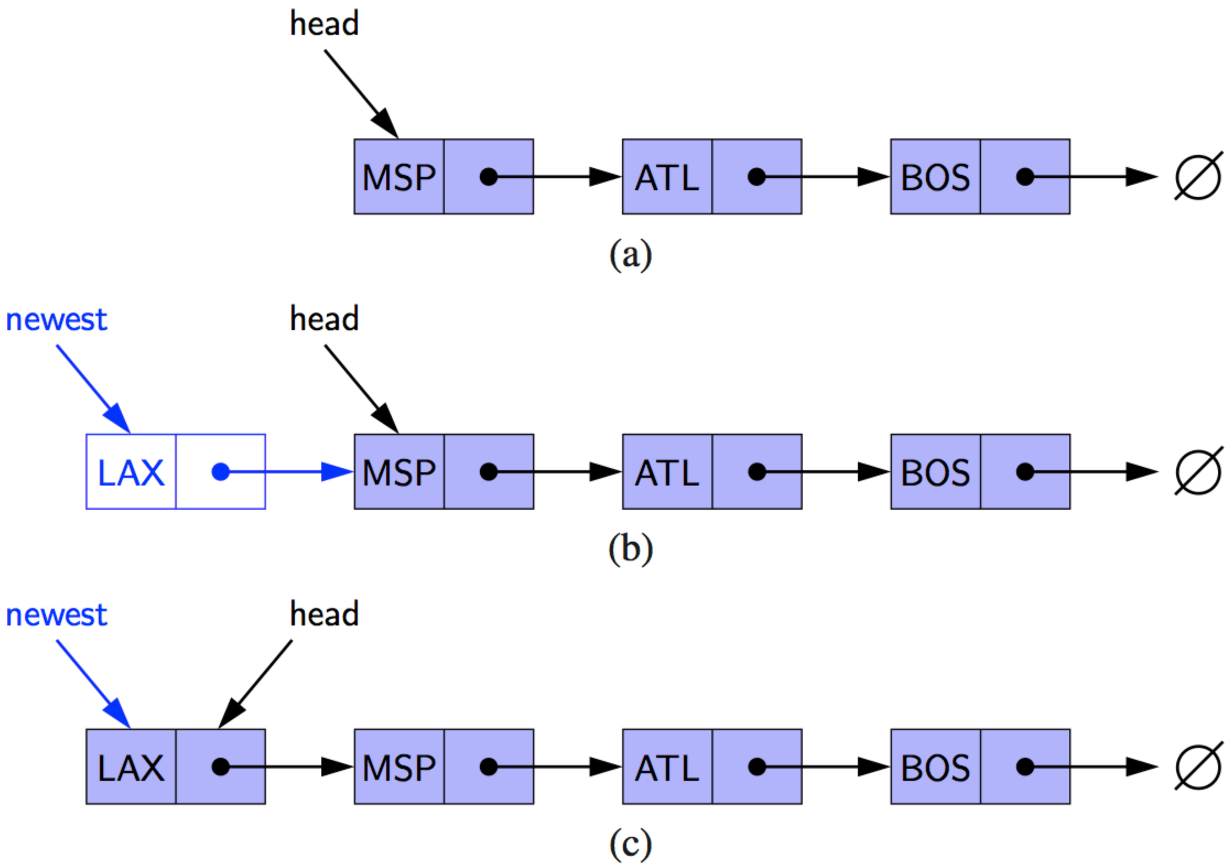
```
public class LinkedListOfRabbits
       implements LinkedListInterface
{
    private class Node
    {
       public Rabbit data;
       public Node next;
       public Node(Rabbit data, Node next)
       {
          this.data = data;
          this.next = next;
       }
    }
    private Node head = null;
    private Node tail = null;
    private int size = 0;
}
```

Print a Linked List

```java
public String toString() {
    StringBuffer s = new StringBuffer();
    for (Node n=head; n!=null; n=n.getNext())
    {
        s.append( n.data.toString());
        if (n != tail)
        {
            s.append("\n");
        }
    }
    return s.toString();
}
```

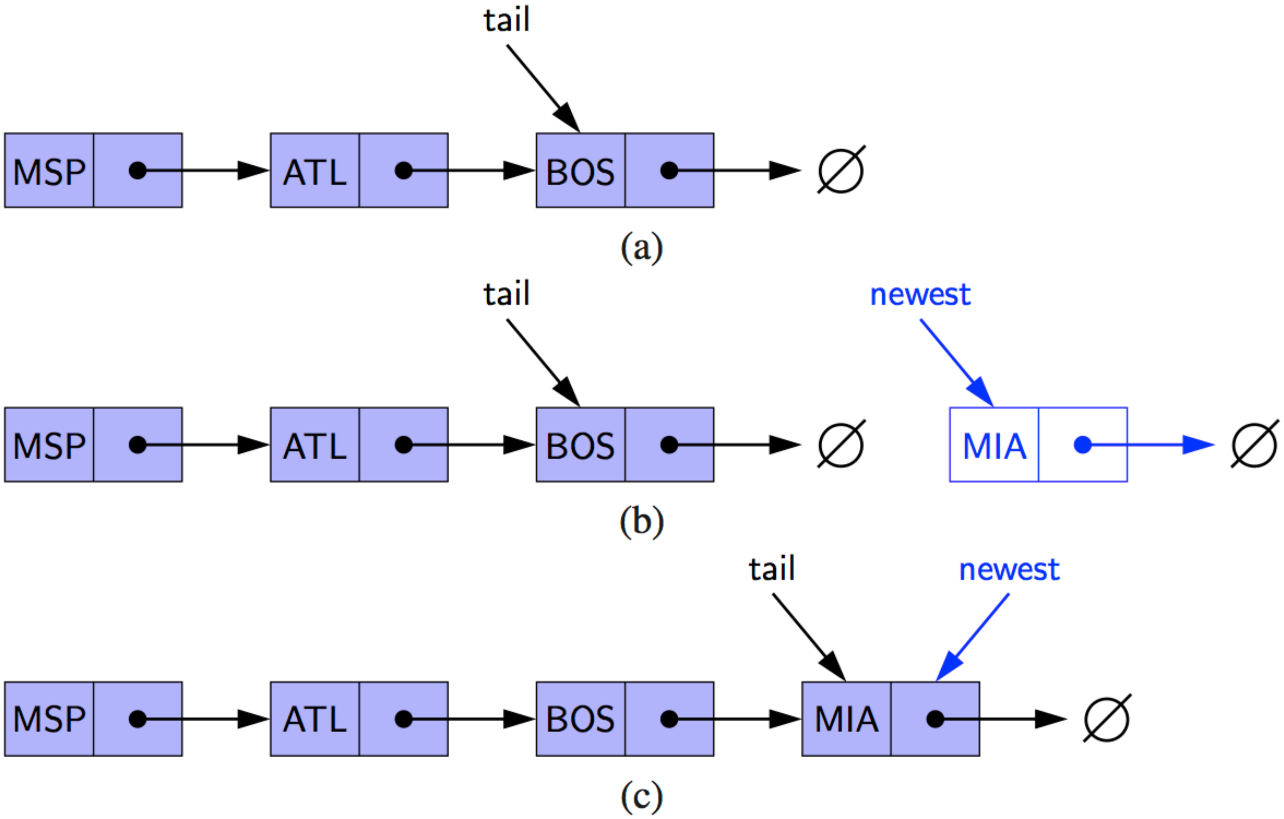
Inserting at the Head

1. create a new node
2. have new node point to old head
3. update head to point to new node



Inserting at the Tail

1. create a new node
2. Have new node point to null
3. have old last node point to new node
4. update tail to point to new node



Insertion

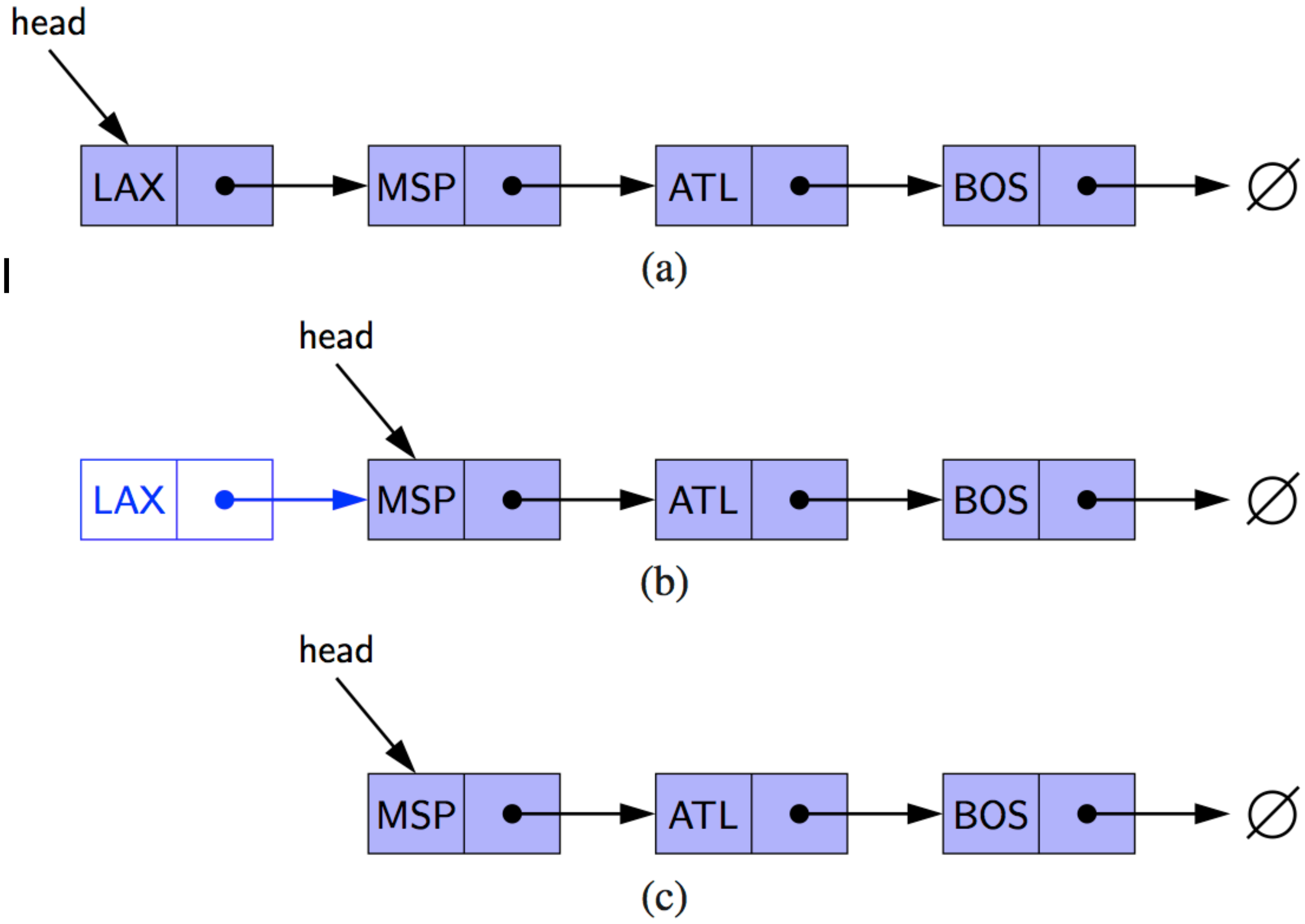
```
public void addLast(Rabbit c)
{
    Node newest = new Node(c, null);
    if (isEmpty())
   { head = newest;}
    else
    {
        tail.next=newest;
    }
    tail = newest;
    size++;
  }
```

Why not take a Node?

write addFirst

Removing at the Head

1. update head to point to next node ii the list
2. allow garbage collector to reclaim the former first node



Deletion

```
public Rabbit removeFirst()
{
   if (isEmpty()) {return null;}
   Rabbit target = head.data;
   head = head.next;
   size--;
   if (isEmpty()) {tail = null;}
   return target;
}
```

Find

```
public Rabbit find(String id)
{
   Node curr = head;
   while (curr!=null)
   {
       if (curr.data.getId().equals(id))
       {
          return curr.data;
       }
        curr=curr.next;
   }
   return null;
}
```