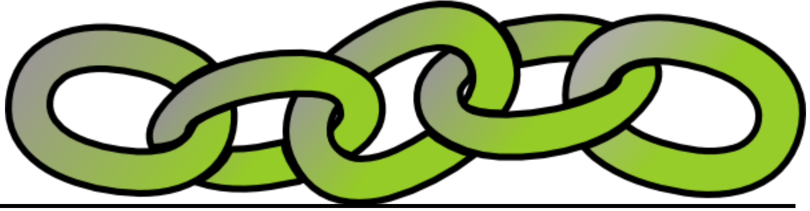
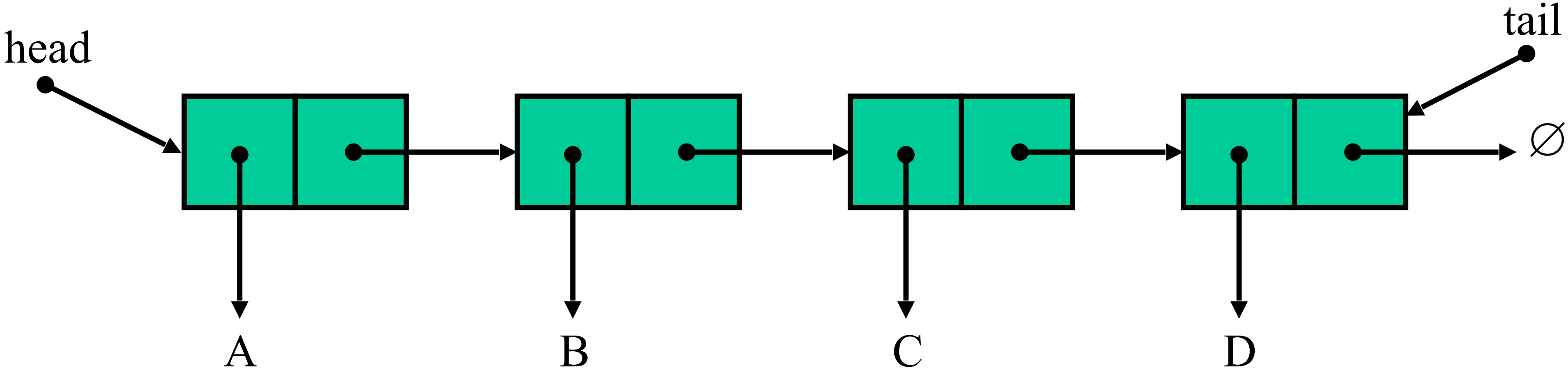
CS206

Linked Lists

Linked List



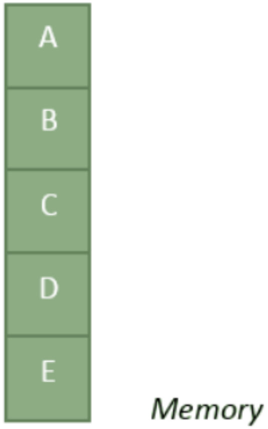
- A linked list is a lists of objects.
- The objects form a linear sequence.
- The sequence is unbounded in length.



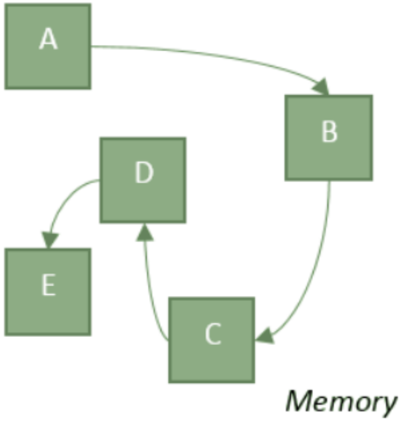
Linked List versus Array

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

Array



Linked List



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)

Linked List Core

- the essential part of a linked list is a “self-referential” structure
- That is, a class with an instance variable that holds a “reference” to another member of that same class
- For linked lists, this structure is usually referred to as a Node

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

References in Java

- A reference variable holds a memory address to where the referenced object is stored (not the object itself)
- Reference types
  - Anything that inherits from Object (including `String`, `Integer`, `Double`, etc)
  - "primitive" types: int, float, etc are NOT reference types
- A reference is `null` when it doesn't refer/ point to any object

References and equality

```
public class ReferenceCheck {
    public static void main(String[] args) {
        String s1 = new String("abc");
        String s2 = new String("abc");
        String s3 = s2;
        String s4 = "abc";

        System.out.println(“s1.equals(s2) " + s1.equals(s2));
        System.out.println("s1==s2 " + (s1 == s2));
        System.out.println("s1==s3 " + (s1 == s3));
        System.out.println("s1==s4 " + (s1 == s4));
        System.out.println("s2==s3 " + (s2 == s3));
        System.out.println("s2==s3 " + (s2 == s4));
        System.out.println("s3==s4 " + (s3 == s4));
    }
}
```

Equals compares content

== compares memory location

Rabbits

You want to store data about a herd of rabbits.

Each rabbit has a breed and birthdate (stored as double) and ID.

Your herd has three breeds, french lop, dwarf dutch, angora

```
  public enum BreedEnum {
      DwarfDutch, Angora, FrenchLop
  }

public class Rabbit {
    private final BreedEnum breed;
    private final String iD;
    public Rabbit(BreedEnum breed, String id) {
        this.breed = breed;
        this.iD = id;
    }
    private Rabbit() {
        this.breed=BreedEnum.Angora;
        this.iD=null;
    }
    public boolean equals(Rabbit otherRabbit) {
        return otherRabbit.getId().equals(this.iD)
    }
    public String getId() { return iD; }
}
```

Enumerated type

Final

Private constructor!

Override equals

Node for Rabbits

```
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!!

handout

Starting Point

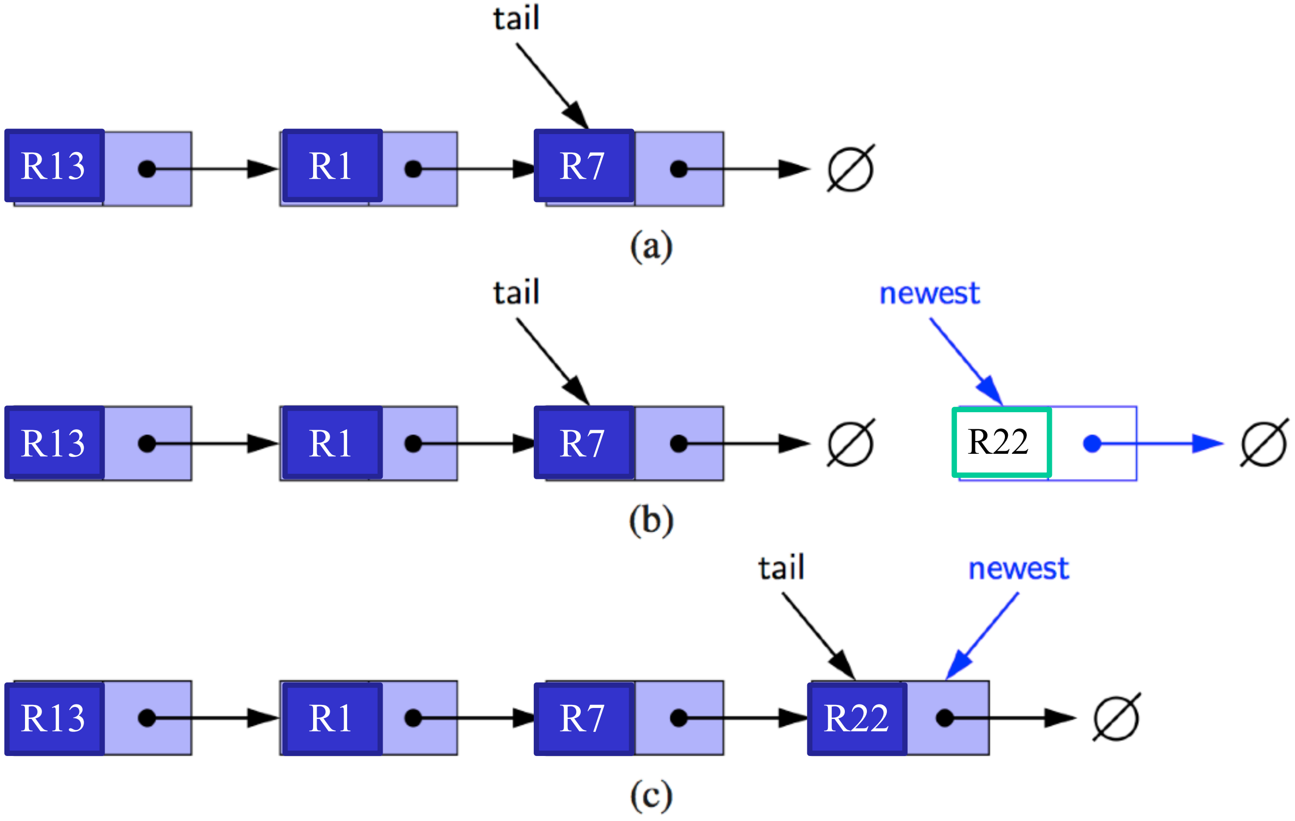
```
public class LinkedListOfRabbits
       implements LinkedListInterface
{
    private class Node
    {
       public Rabbit data;
       public Node next;
       public Node(Rabbit data, Node next)
       {
          this.data = data;
          this.next = next;
       }
    }
    private Node head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

Print a Linked List

```
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 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



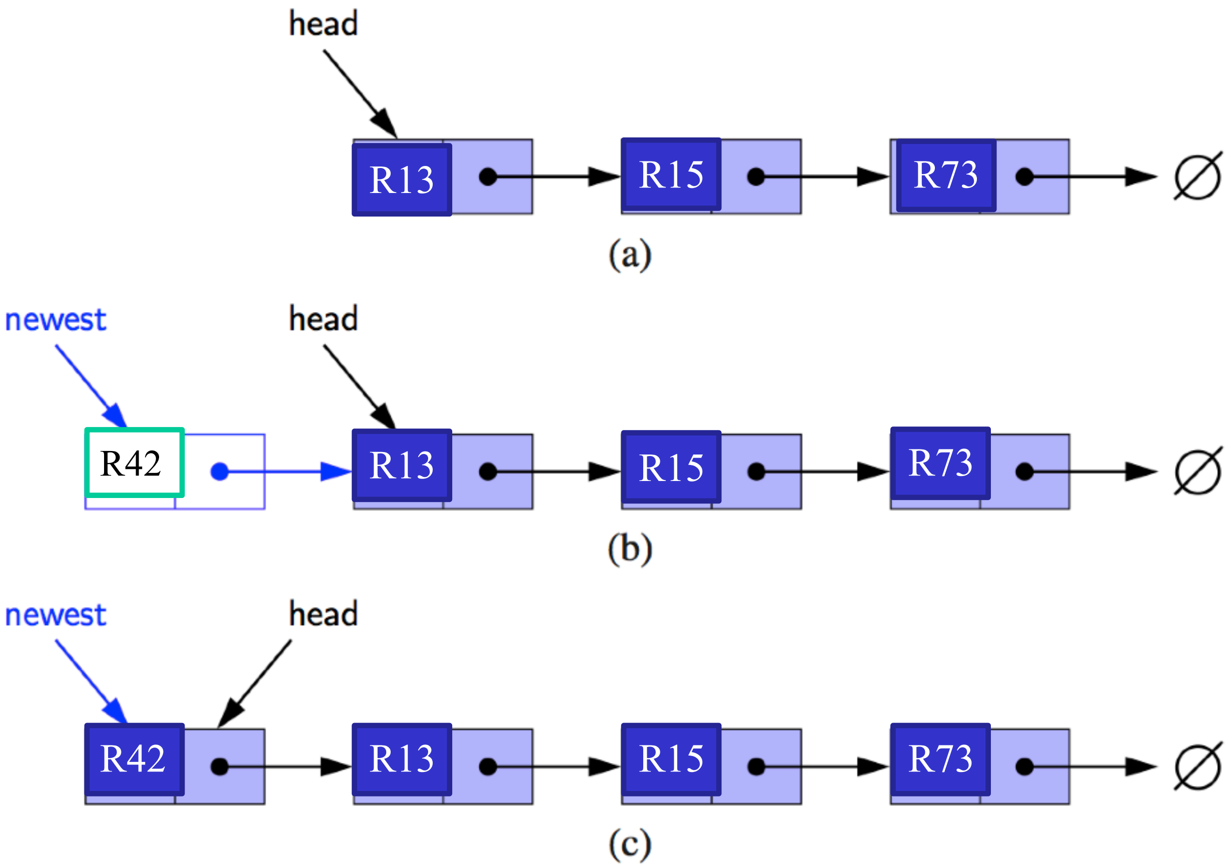
Insertion

```
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?

Inserting at the Head

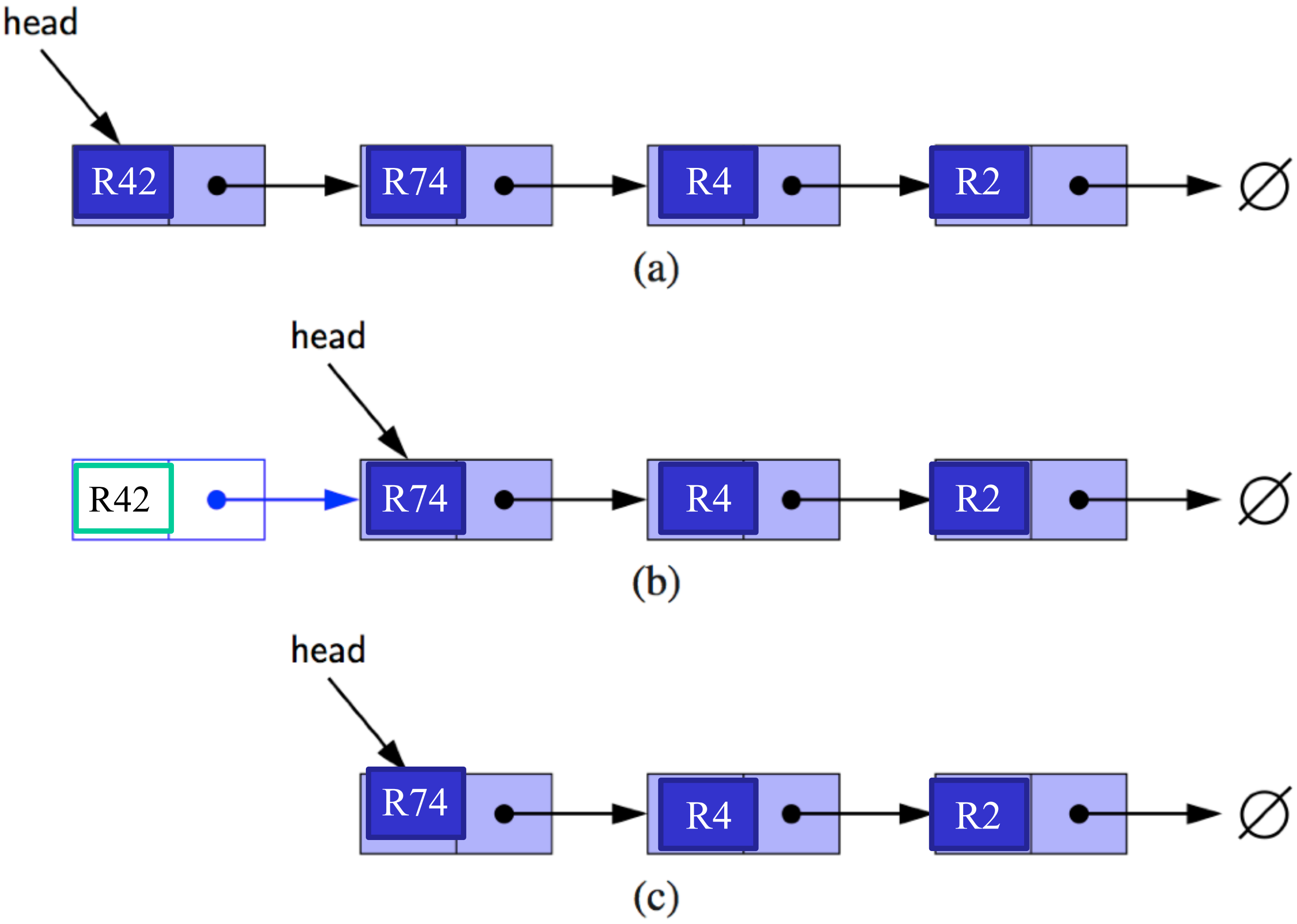
1. create a new node
2. have new node point to old head
3. update head to point to new node



write addFirst at chalkboard

Removing at the Head

1. update head to point to next node in 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