Lecture 16 - Wednesday, March 8

Announcements

- ProgTest1 results to be released by Friday, March 17
- Makeup Lecture for WrittenTest1, ProgTest1
  + Expected to complete by: March 20

Implementing the Queue ADT using a SLL

Exercise

(1) DLL, first is front of Q

(2) DLL, last is front of Q.

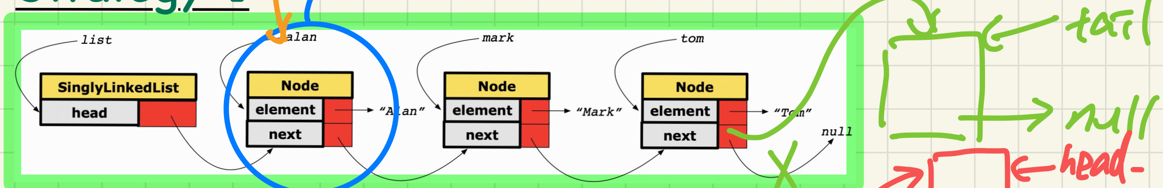
```java
public class LinkedQueue<E> implements Queue<E> {
  private SinglyLinkedList<E> list;
  ...
}
```

Queue Method	Singly-Linked List Method: Strategy 1	Singly-Linked List Method: Strategy 2
size	list.size	list.size
isEmpty	list.isEmpty	list.isEmpty
first	list.first O(1)	list.last O(1)
enqueue	✓ list.addLast O(1)	list.addFirst O(1)
dequeue	list.removeFirst O(1)	list.removeLast O(n)

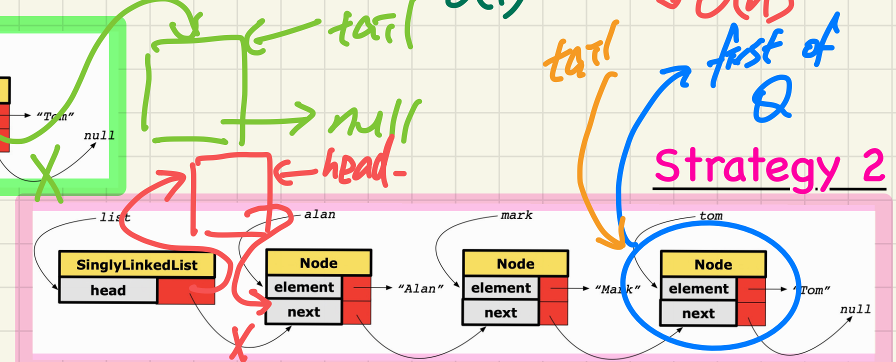
head

first of Q.

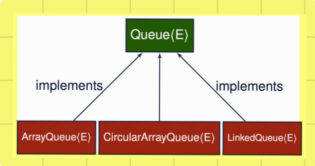
Strategy 1



Strategy 2



Queue ADT: Testing Alternative Implementations



Polymorphism.

```java
@Test
public void testPolymorphicQueues() {
  Queue<String> q = new ArrayQueue<>();
  q.enqueue("Alan"); /* dynamic binding */
  q.enqueue("Mark"); /* dynamic binding */
  q.enqueue("Tom"); /* dynamic binding */
  assertTrue(q.size() == 3 && !q.isEmpty());
  assertEquals("Alan", q.first());

  q = new LinkedQueue<>();
  q.enqueue("Alan"); /* dynamic binding */
  q.enqueue("Mark"); /* dynamic binding */
  q.enqueue("Tom"); /* dynamic binding */
  assertTrue(q.size() == 3 && !q.isEmpty());
  assertEquals("Alan", q.first());
}
```

dynamic binding.

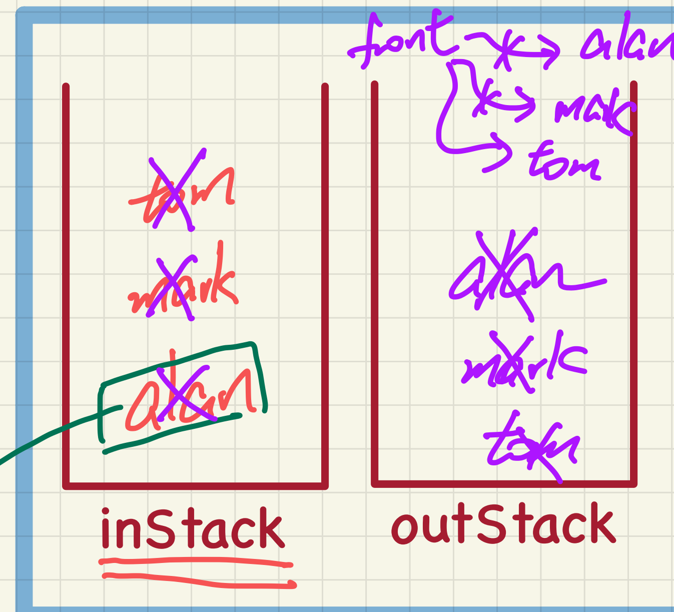
```java
public class ArrayQueue<E> implements Queue<E> {
  private final int MAX_CAPACITY = 1000;
  private E[] data;
  private int r = -1; /* rear index */
  public ArrayQueue() {
    data = (E[]) new Object[MAX_CAPACITY];
    r = -1;
  }
  public int size() { return (r + 1); }
  public boolean isEmpty() { return (r == -1); }
  public E first() {
    if (isEmpty()) { /* Precondition Violated */ }
    else { return data[0]; }
  }
  public void enqueue(E e) {
    if (size() == MAX_CAPACITY) { /* Precondition Violated */ }
    else { r ++; data[r] = e; }
  }
  public E dequeue() {
    if (isEmpty()) { /* Precondition Violated */ }
    else {
      E result = data[0];
      for (int i = 0; i < r; i ++) { data[i] = data[i + 1]; }
      data[r] = null; r --;
      return result;
    }
  }
}
```

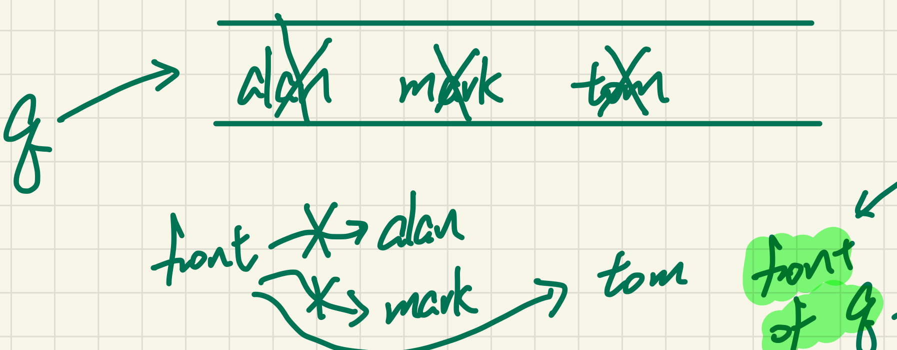
Exercise: Implementing a Queue using Two Stacks

Queue Operation:

```
q.enqueue("alan");
q.enqueue("mark");
q.enqueue("tom");
String front = q.dequeue();
front = q.dequeue();
front = q.dequeue();
```

when the ① dequeue is demanded and ② outStack is empty





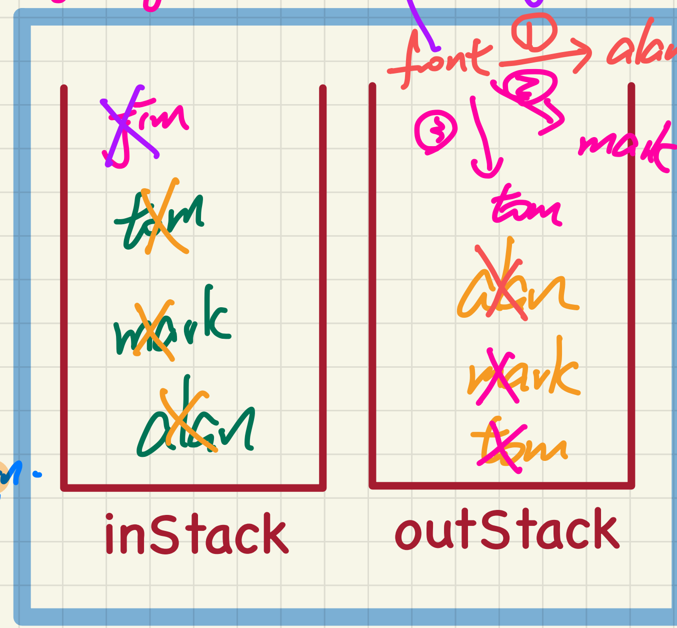
Queue Operation:

```
q.enqueue("alan");
q.enqueue("mark");
q.enqueue("tom");
String front = q.dequeue();
front = q.dequeue();
front = q.dequeue();
```

q.dequeue();

LIFO vs FIFO

q.enqueue("Jim");



top of outStack

Only pop everything off "inStack" and push to "outStack" if:

(1) a "front" or "dequeue" demanded.

(2) "outStack" is empty.

Lecture

General Trees ADT

Terminology, Applications

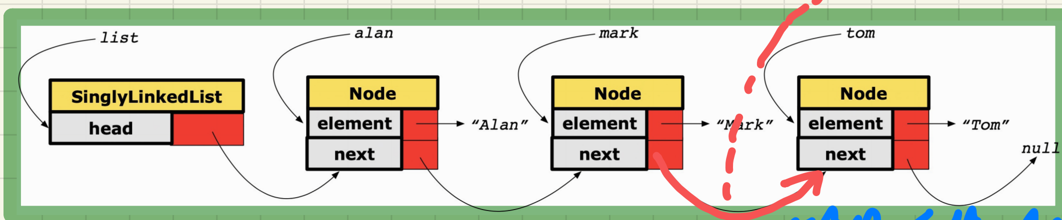
Trees

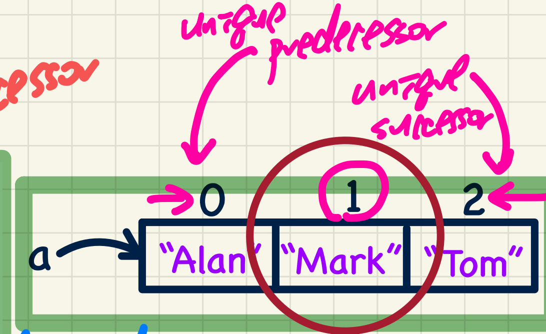
a.
1. General Trees
2. Binary Trees (BTs)

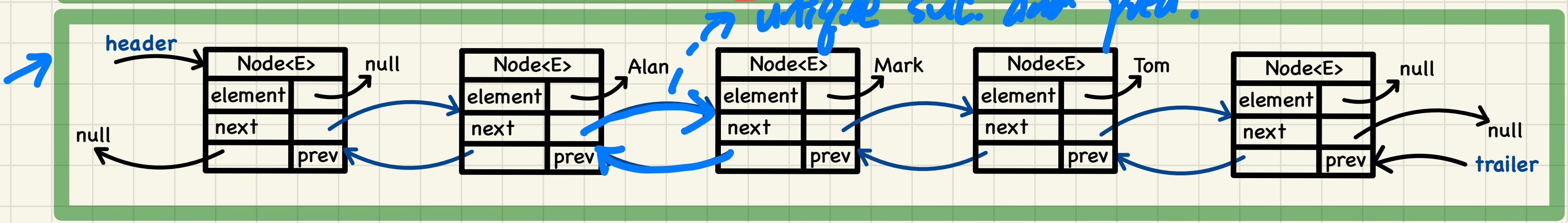
b.
3. Binary Search Trees (BSTs)

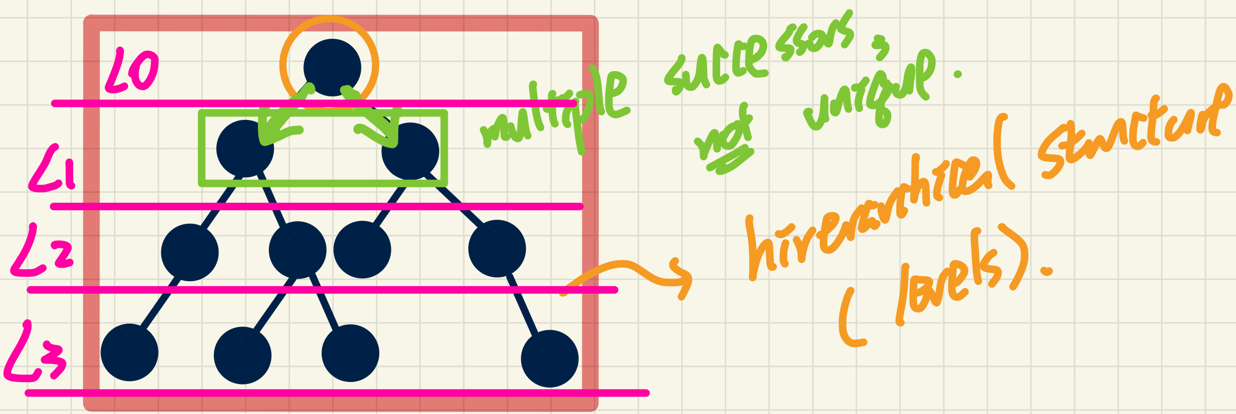
c.
4. Balanced BSTs
5. ADT: Priority Queues
6. Heap Sort

Linear vs. Non-Linear Structures

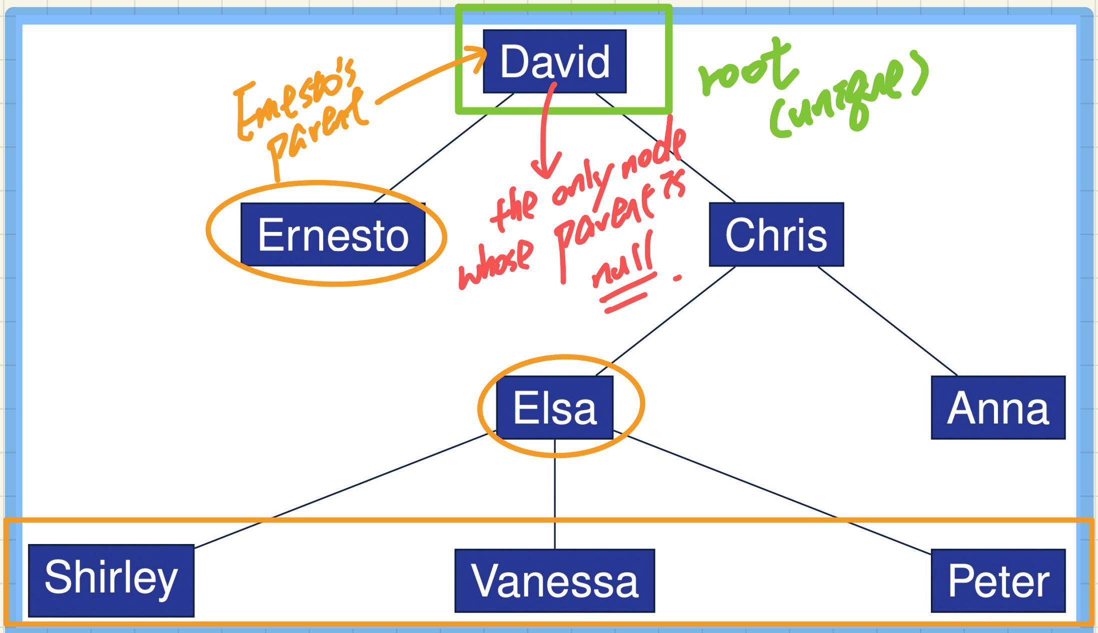








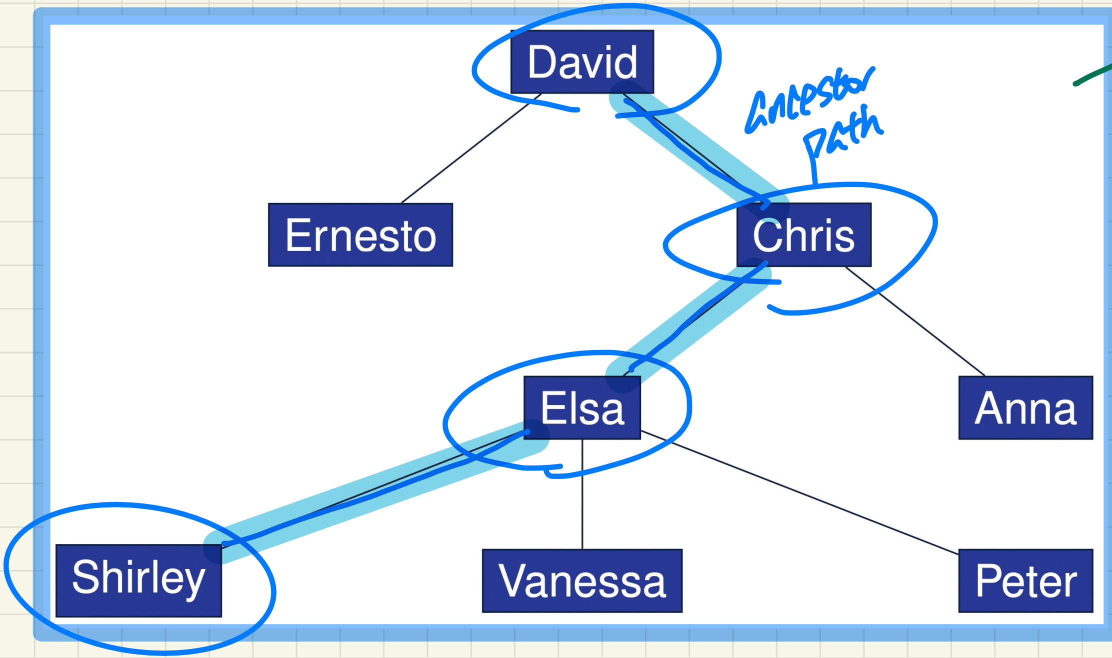
General Trees: Terminology (1)



children of Elsa

- root
- parent — immediately above node
- children
- ancestors
- descendants
- siblings

siblings: nodes sharing the same parents: e.g. Ernesto, Chris



Every node has a unique parent.

A node is both its ancestor and descendant.

Ancestors of Shirley : Shirley, Elsa, Chris, David.

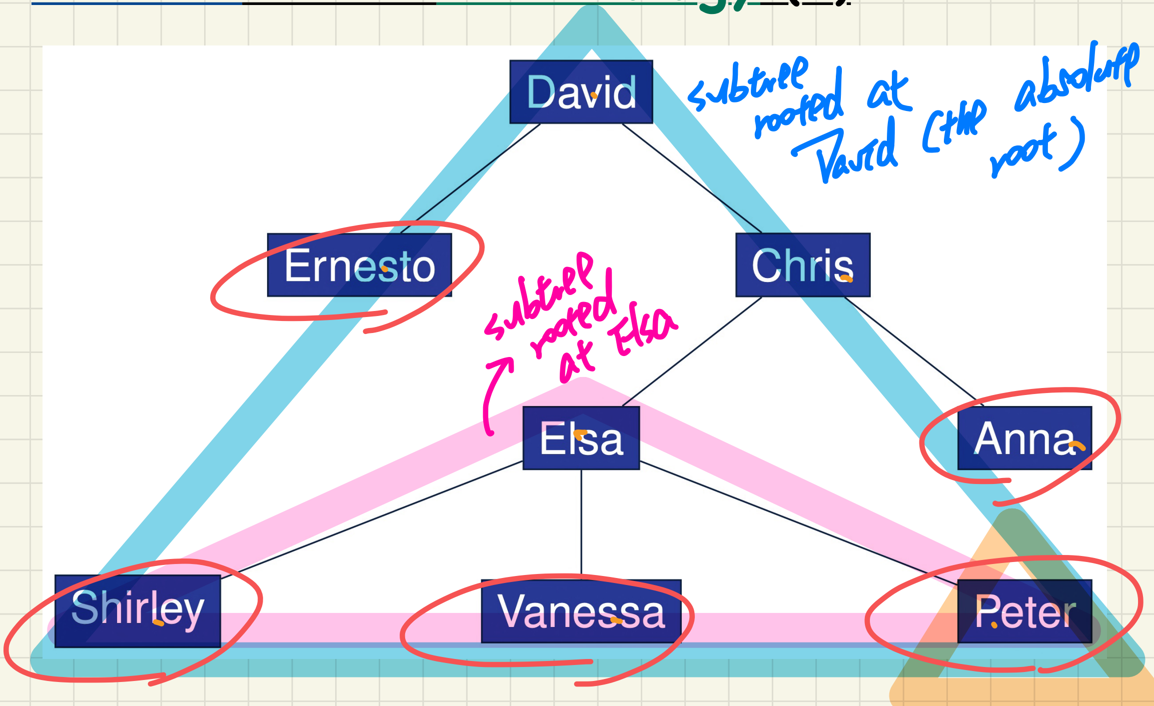
Descendants of the root cover the entire tree.

Descendants of Ernesto : Ernesto

Descendants of Chris : C, Elsa, Anna, S, V, P.

General Trees: Terminology (2)

- subtree



subtree rooted at Peter

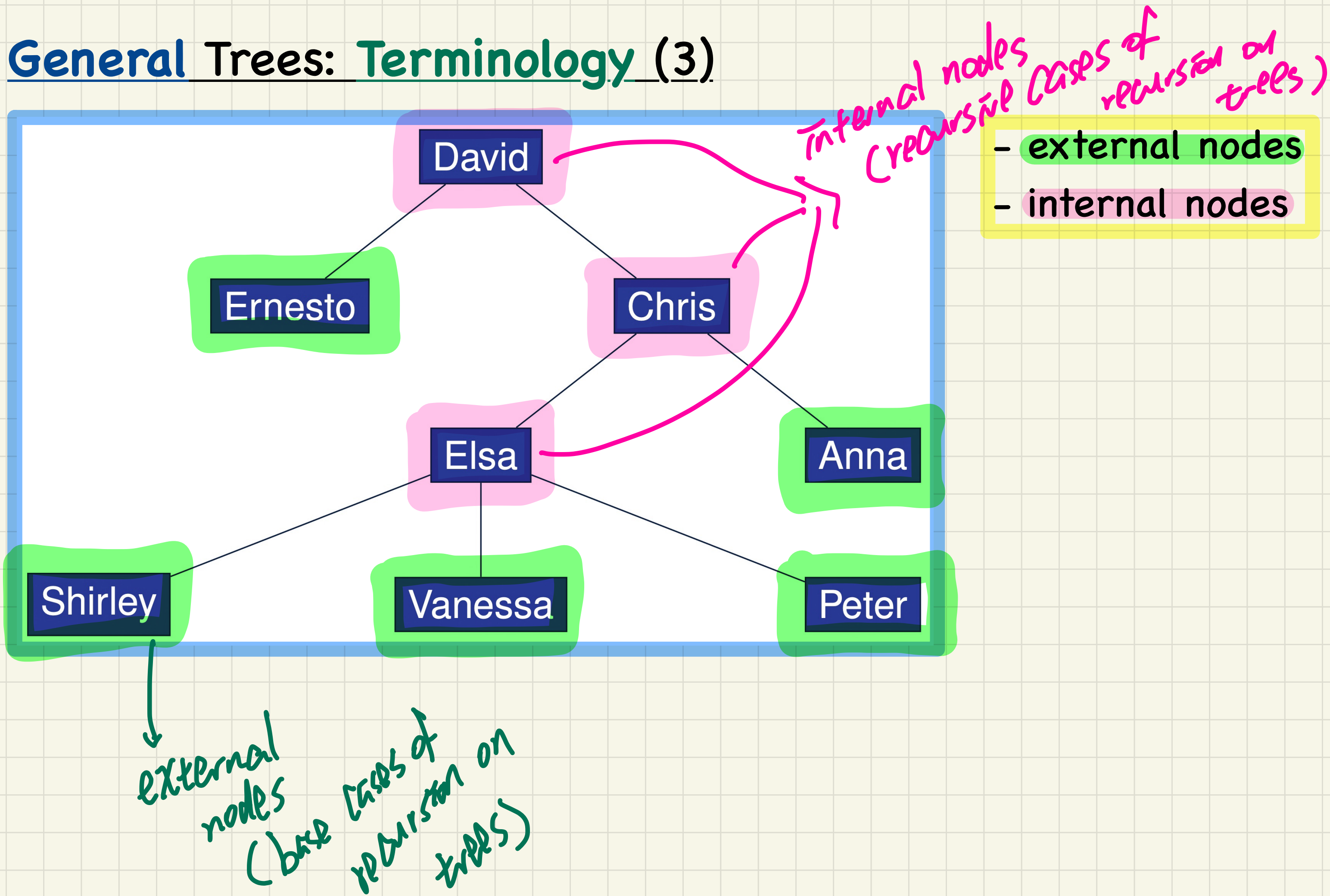
How many subtrees are there in the tree?

8 (# of nodes in the tree).

size \ ST	
1	5 STs ...
2	
⋮	
8	

- subtree rooted at David.
- subtree rooted at Peter.
- subtree rooted at Elsa.

General Trees: Terminology (3)



- external nodes
- internal nodes