Tree Traversal

- Traversing a tree = visiting every node in the tree

- Depth-first traversal:
  - Follows all the way down one subtree before starting other subtree(s)
  - Easy to do with recursion.

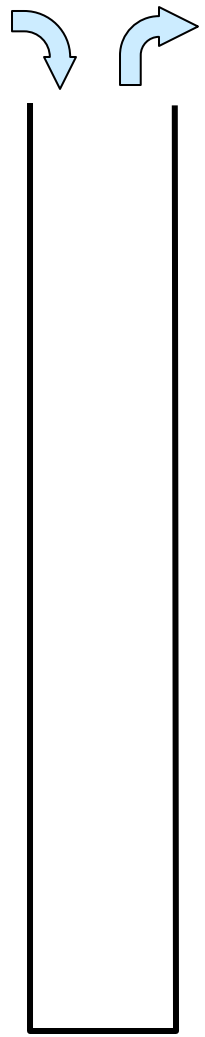
- For Binary trees (two children per node)
  - pre-order: visit parent node then traverse child subtrees,
  - post-order: traverse child subtrees then visit parent node
  - in-order: traverse one child subtree
    then visit parent node
    then traverse other subtree
    - (binary trees only)

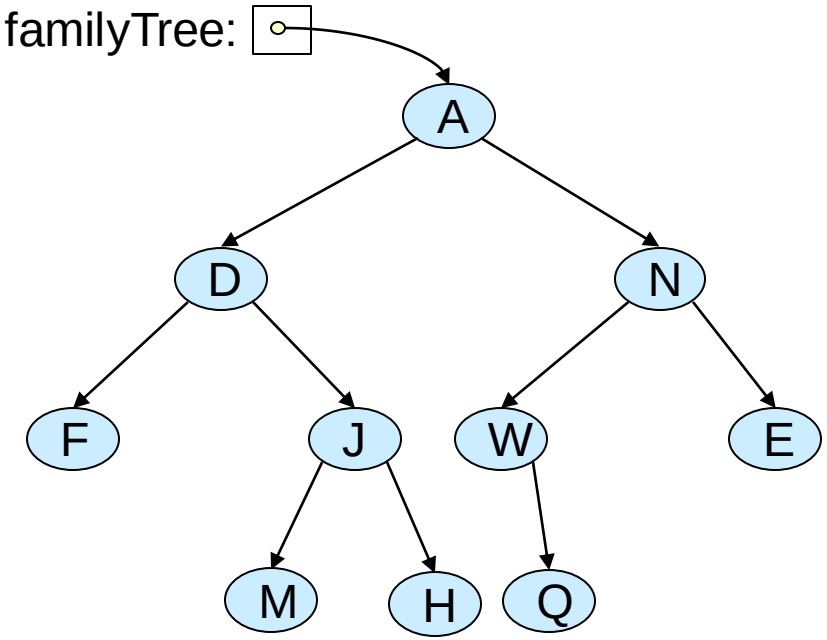
© Peter Andreae

Depth first traversal without recursion

- Use a stack to store the nodes that need to be worked on

```
public void preOrderDF (Person root){
    Stack<Person> todo = new Stack<Person>();
    todo.push(root);
    while ( ! todo.isEmpty() ){
        Person p = todo.pop()
        UI.println(p);
        if ( p.getMother() != null ){
            todo.push(p.getMother());
        }
        if ( p.getFather() != null ){
            todo.push(p.getFather());
        }
    }
}
```



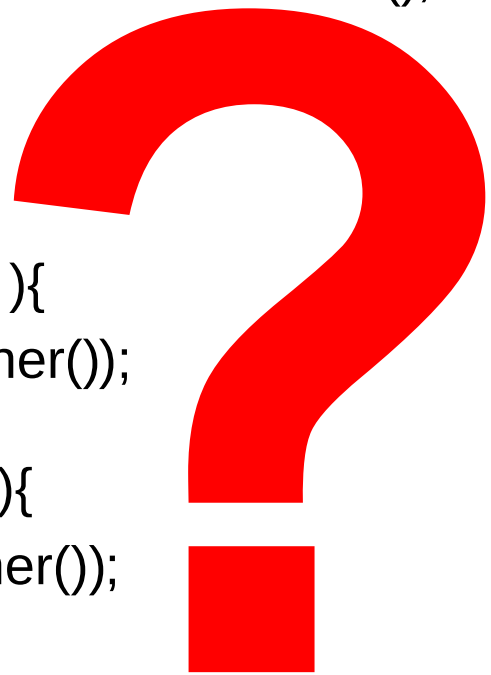


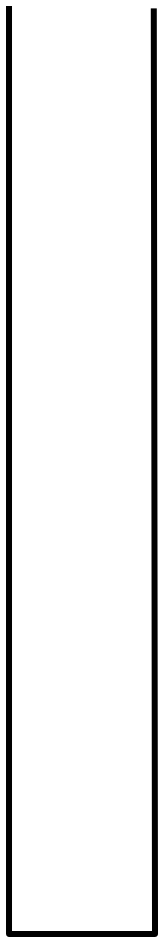
© Peter Andreae

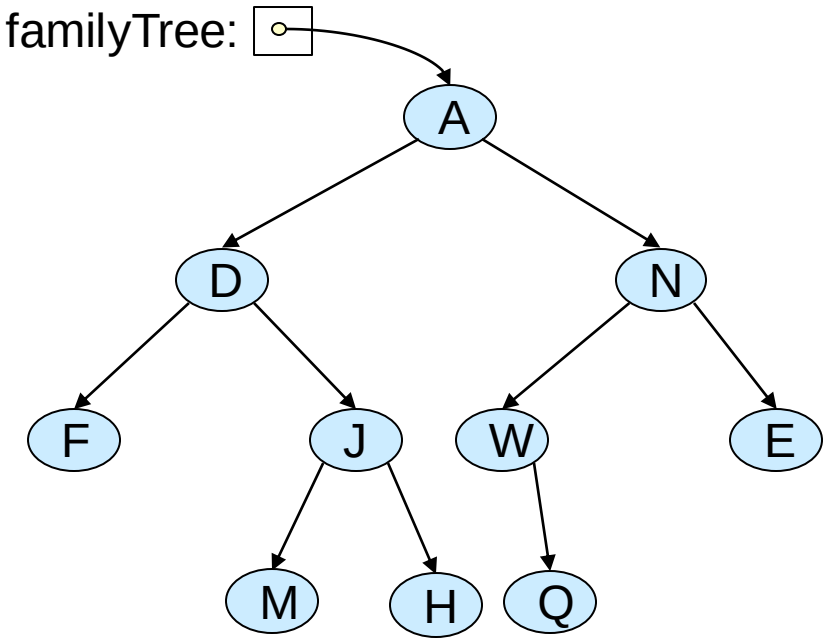
Depth first traversal without recursion

- How do we do post-order?

```
public void preOrderDF (Person root){
    Stack<Person> todo = new Stack<Person>();
    todo.push(root);
    while ( ! todo.isEmpty() ){
        Person p = todo.pop()
        if ( p.getMother() != null ){
            todo.push(p.getMother());
        }
        if ( p.getFather() != null ){
            todo.push(p.getFather());
        }
        UI.println(p);
    }
}
```



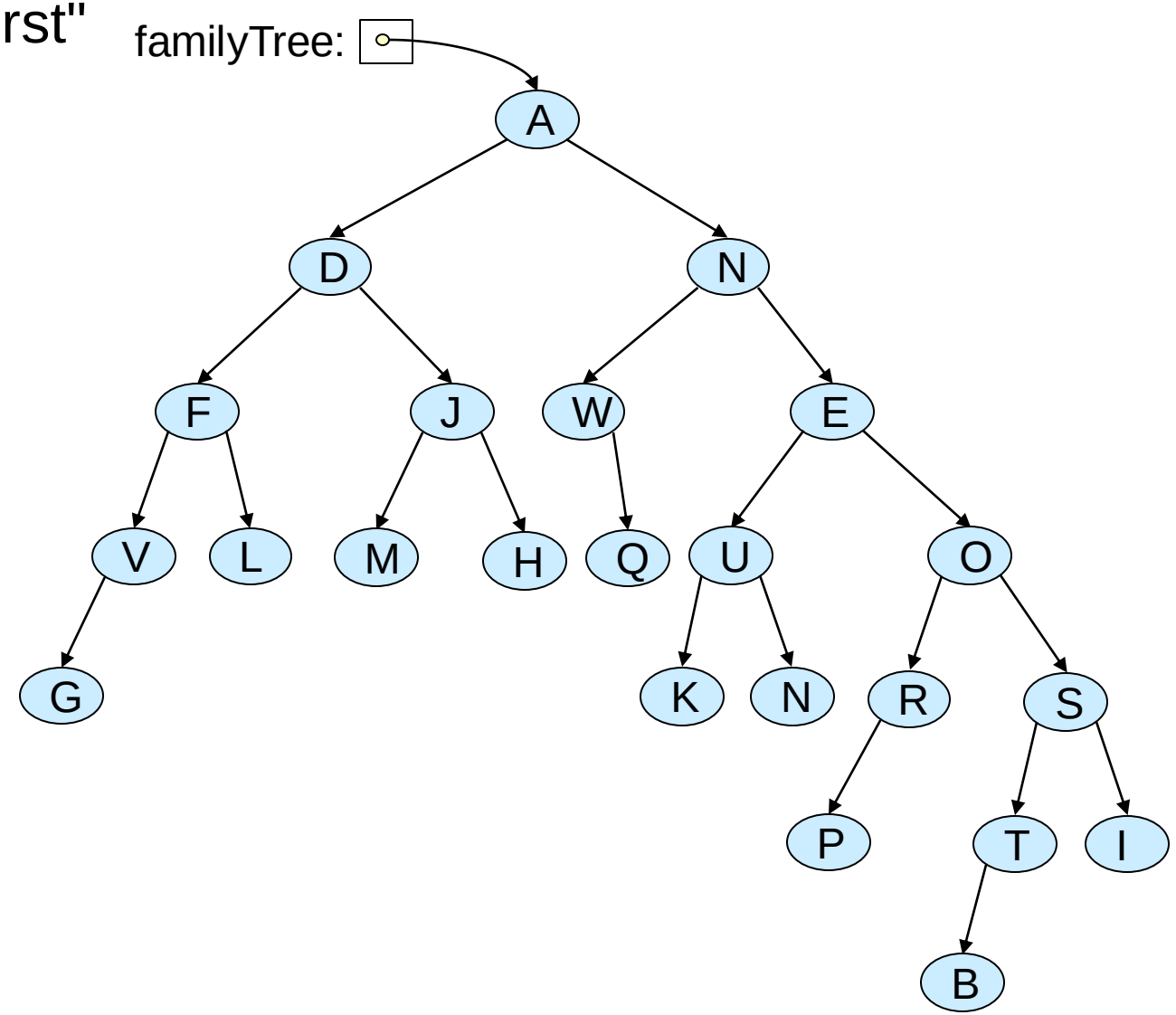




© Peter Andreae

Breadth First Traversal

- Traversing nodes by level = "breadth first"

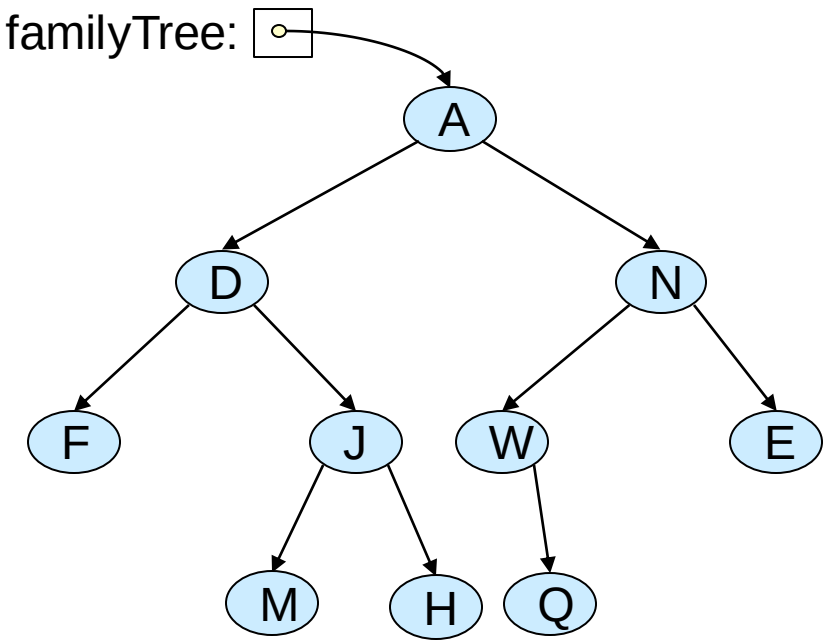


© Peter Andreae

Breadth first

- Use a stack to store the nodes that need to be worked on

```
public void breadthFirstTraversal (Person root){
    Queue<Person> todo = new ArrayDeque<Person>();
    todo.offer(root);
    while ( ! todo.isEmpty() ){
        Person p = todo.poll();
        UI.println(p);
        if ( p.getMother() != null ){
            todo.offer(p.getMother());
        }
        if ( p.getFather() != null ){
            todo.offer(p.getFather());
    }
}
```



© Peter Andreae

Traversing and returning

- Collecting up nodes in a list/set to return:
  - Straightforward using the iterative (Stack or Queue based) traversal:

```
/** Find all Persons in tree born before a given year */
  public List<Person> dfFindOldStack(Person root, int year){
    List<Person> ans = new ArrayList<Person>();
    Stack<Person> todo = new Stack<Person>();
    todo.push(root);
    while ( ! todo.isEmpty() ){
        Person p = todo.pop();
        if (p.getYoB()< year)      { ans.add(p); }
        if ( p.getMother() != null ){ todo.push(p.getMother()); }
        if ( p.getFather() != null ) { todo.push(p.getFather());  }
    }
    return ans;
  }
```

© Peter Andreae

Traversing and returning

- Collecting up nodes in a list/set to return:
  - In recursive traversal, pass in List/Set; method just adds values to List/Set;
  - No need to return list from recursive calls

```
/** Find all Persons in tree born before a given year */
public List<Person> dfFindOldRec(Person p, int year){
    List<Person> listOfOld = new ArrayList<Person>();
    dfFindOldRecHelper(p, year, listOfOld);
    return listOfOld;
}
public void  dfFindOldRecHelper(Person p, int year, List<Person> listOfOld){
    if (p!=null){
        if (p.getYoB()< year) { listOfOld.add(p); }
        dfFindOldRecHelper(p.getFather(), year, listOfOld);
        dfFindOldRecHelper(p.getMother(), year, listOfOld);
    }
}
```

© Peter Andreae

Traversing and returning

- Finding a single node or value to return:
  - Straightforward using the iterative (Stack or Queue based) traversal:

```
/** Find a Person in tree with a given name */
  public Person dfFindNameStack(Person root, String name){
     Stack<Person> todo = new Stack<Person>();
     todo.push(root);
     while ( ! todo.isEmpty() ){
        Person p = todo.pop();
        if (p.getName().equals(name))  { return p; }
        if ( p.getMother() != null ){ todo.push(p.getMother()); }
        if ( p.getFather() != null ) { todo.push(p.getFather());  }
     }
    return null;
  }
```

© Peter Andreae

Traversing and returning

- Finding a single node or value to return:
  - In recursive traversal, must pass back the answer, all the way up the tree

```
/** Find a Person in tree with a given name */
public Person dfFindNameRec(Person p, String name){
    if (p!=null){
        if (p.getName().equals(name)) {return p;}
        Person ans = dfFindNameRec(p.getFather(), name);
        if (ans !=null) { return ans; }
        return dfFindNameRec(p.getMother(), name);
    }
}
```

© Peter Andreae