Data Structures and Algorithms

XMUT-COMP 103 - 2026 T1

General Trees

Agatha Rachmat

School of Engineering and Computer Science

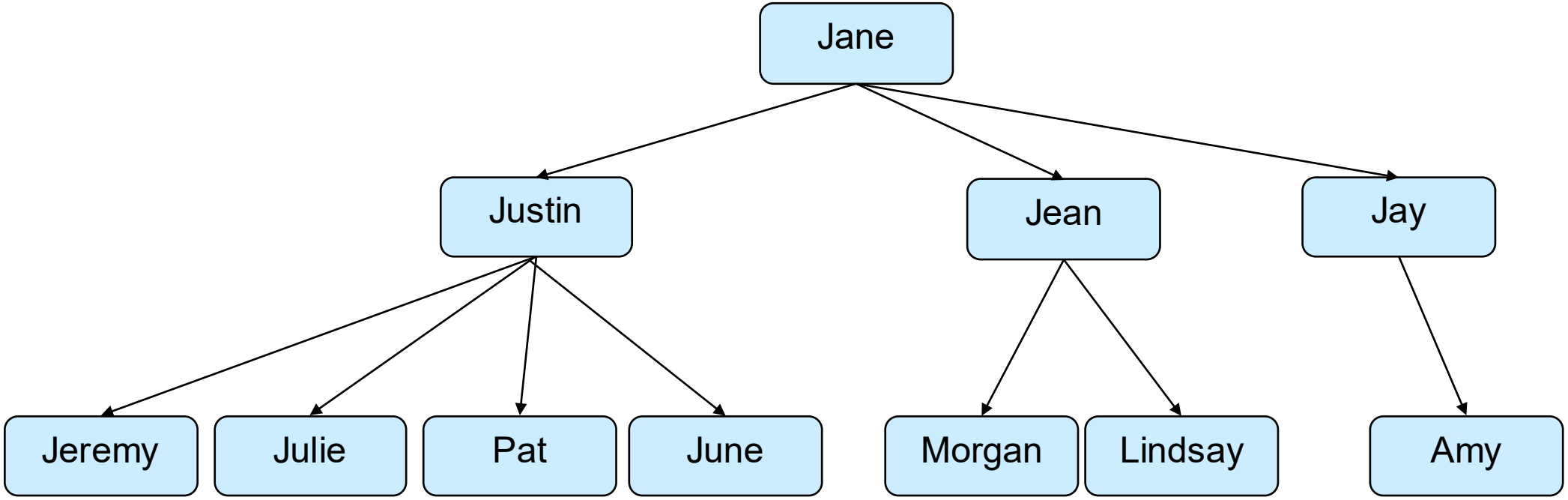
Victoria University of Wellington

Admin

Mid-test result is out. Please check

General Trees

- Binary Trees : at most two child nodes.
- Ternary Trees : at most three child nodes.
- General Trees : any number of child nodes.

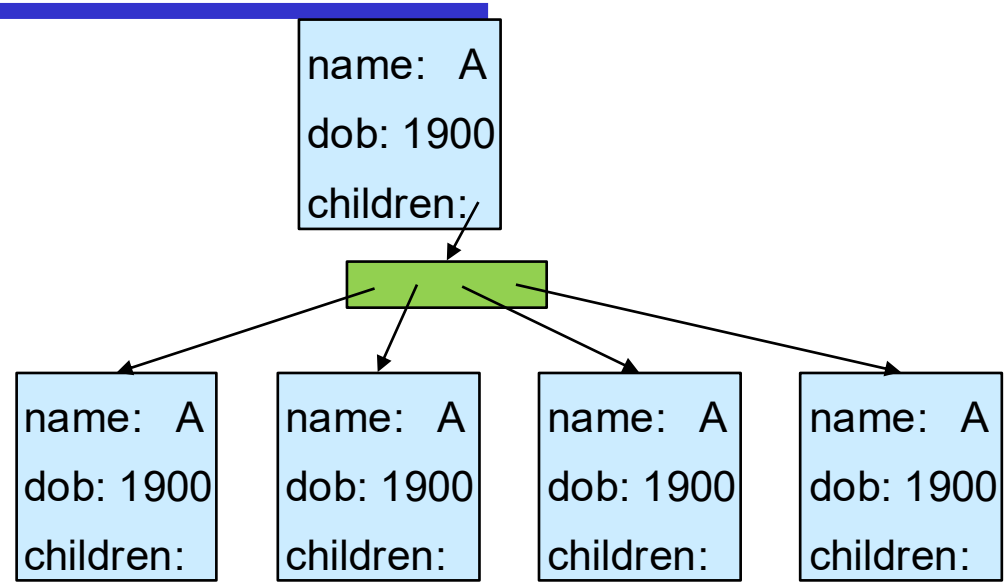


Data Structures for General Trees

```
public class Person {
    private String name;
    private int dob;
    private Set<Person> children;

        :
    public Set<Person> getChildren(){
        return Collections.unmodifiableSet(children);
    }
    public void addChild(Person ch){
        children.add(ch);
    }
    public void removeChild(Person ch){
        children.remove(ch);
    }

}
```

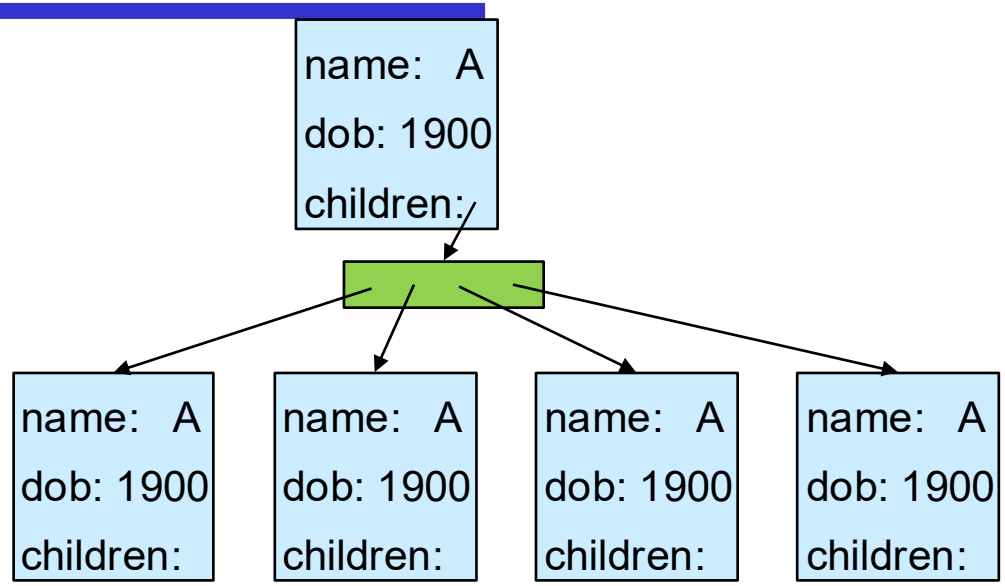


Safer:
returns set of children, but the set can't be modified

removes the child AND its subtree

General Trees

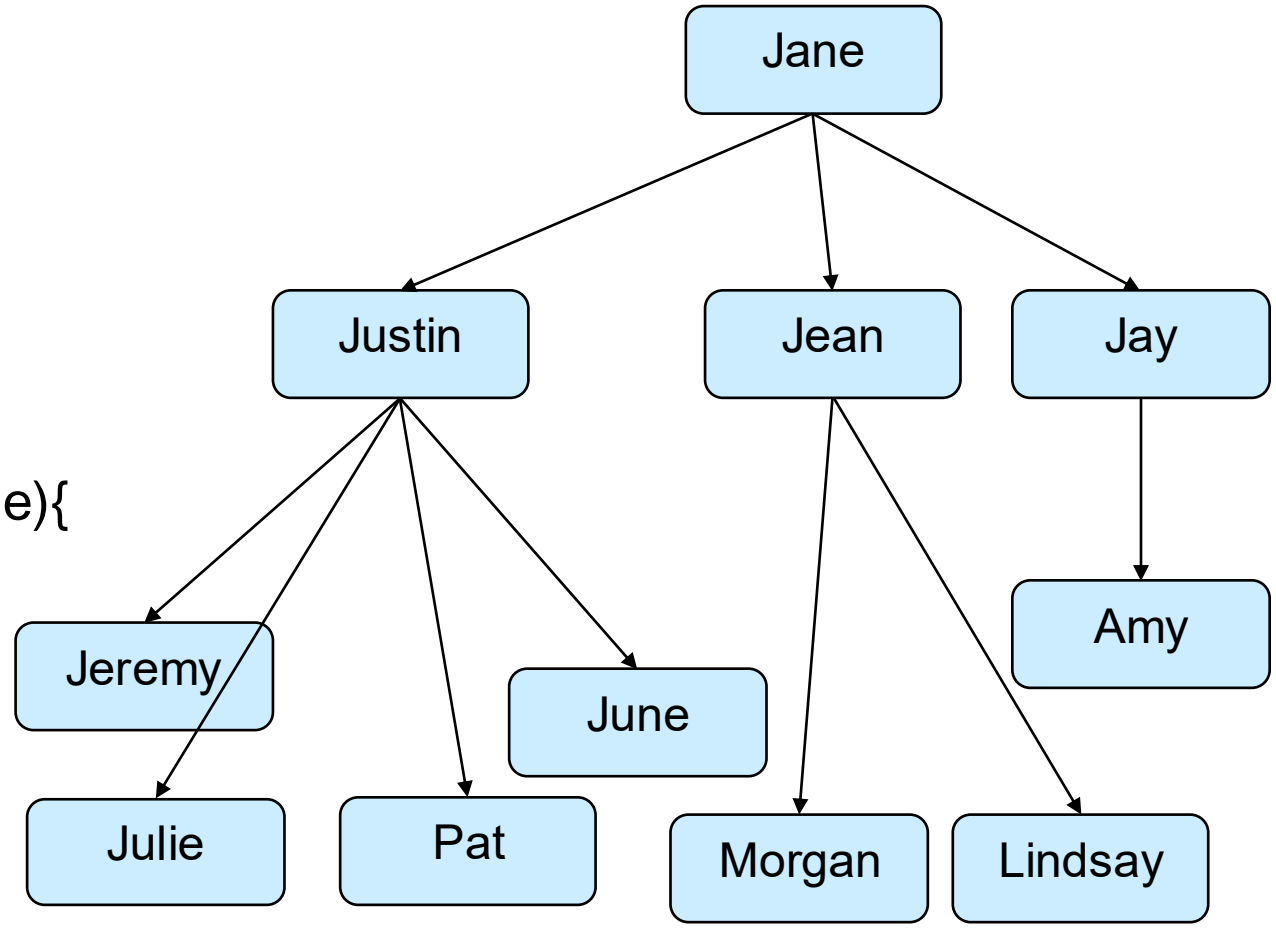
- Depth-First with Recursive
- Depth-First Traversal with a Stack
- Breadth-first



Traversing General Trees

Recursive Depth first traversal; just like binary trees:

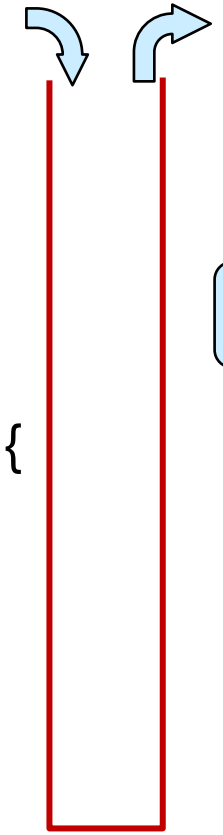
```
public void printTree(Person p){
    if (p==null) { return; }
    UI.println(p);
    for (Person child : p.getChildren(){
        printTree(child);
    }
}
public Person findPerson(Person p, String name){
    if (p==null) { return null; }
    if (p.getName().equals(name)) { return p; }
    for (Person child : p.getChildren(){
        Person ans = findPerson(child, name);
        if (ans!=null) { return ans; }
    }
    return null;
}
```

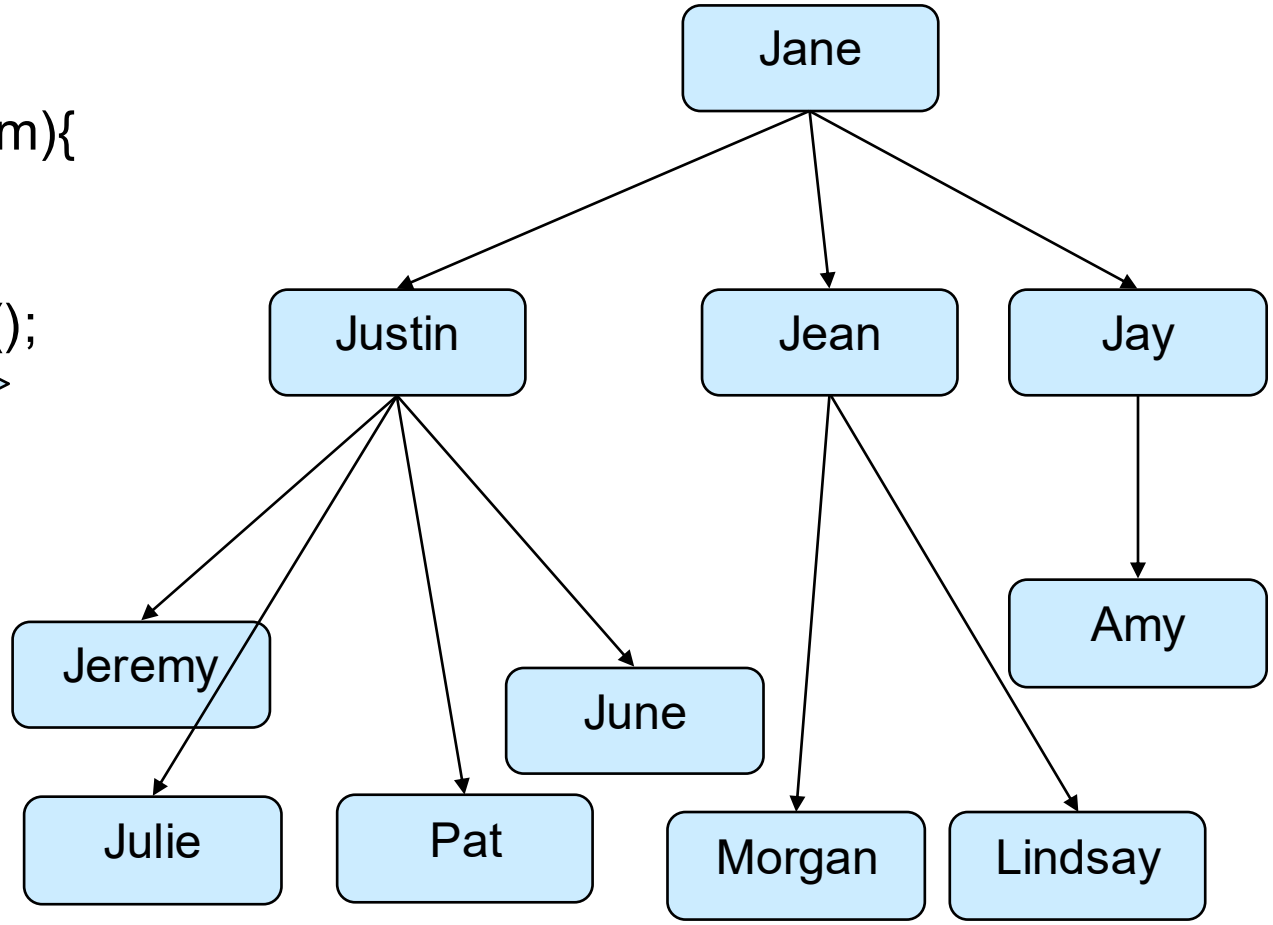


Depth-First Traversal with a Stack

- Traversing the tree by level

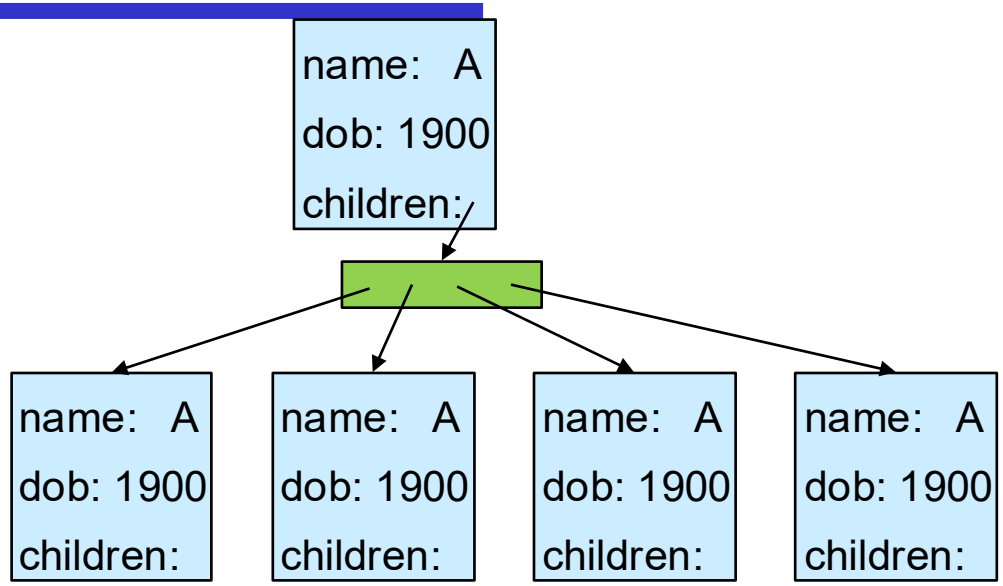
```
public Person findPerson (Person root, String nm){
    if (root==null || nm == null) { return null; }
    Stack<Person> todo = new Stack<Person>();
    todo.push(root);
    while (!todo.isEmpty()){
        Person p = todo.pop ();
        if (p.getName().equals(nm)){
            return p;
        }
        for (Person ch : p.getChildren()){
            todo.push(ch);
        }
    }
    return null;
}
```





General Trees

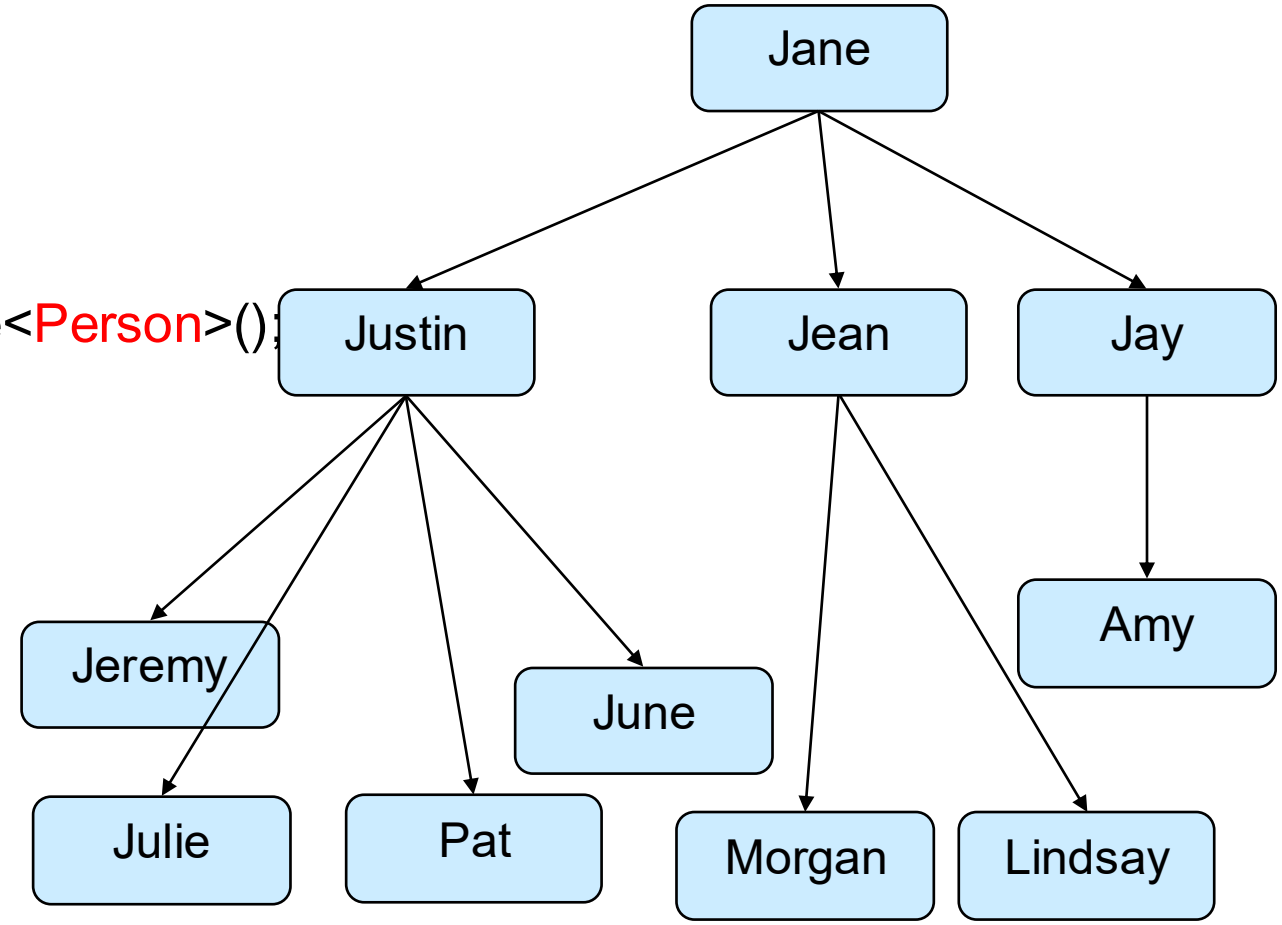
- Depth-First with Recursive
- Depth-First Traversal with a Stack
- **Breadth-first**
  - Visit the root first
  - Then visit all children of the root
  - Then all grandchildren



Breath-First Traversal

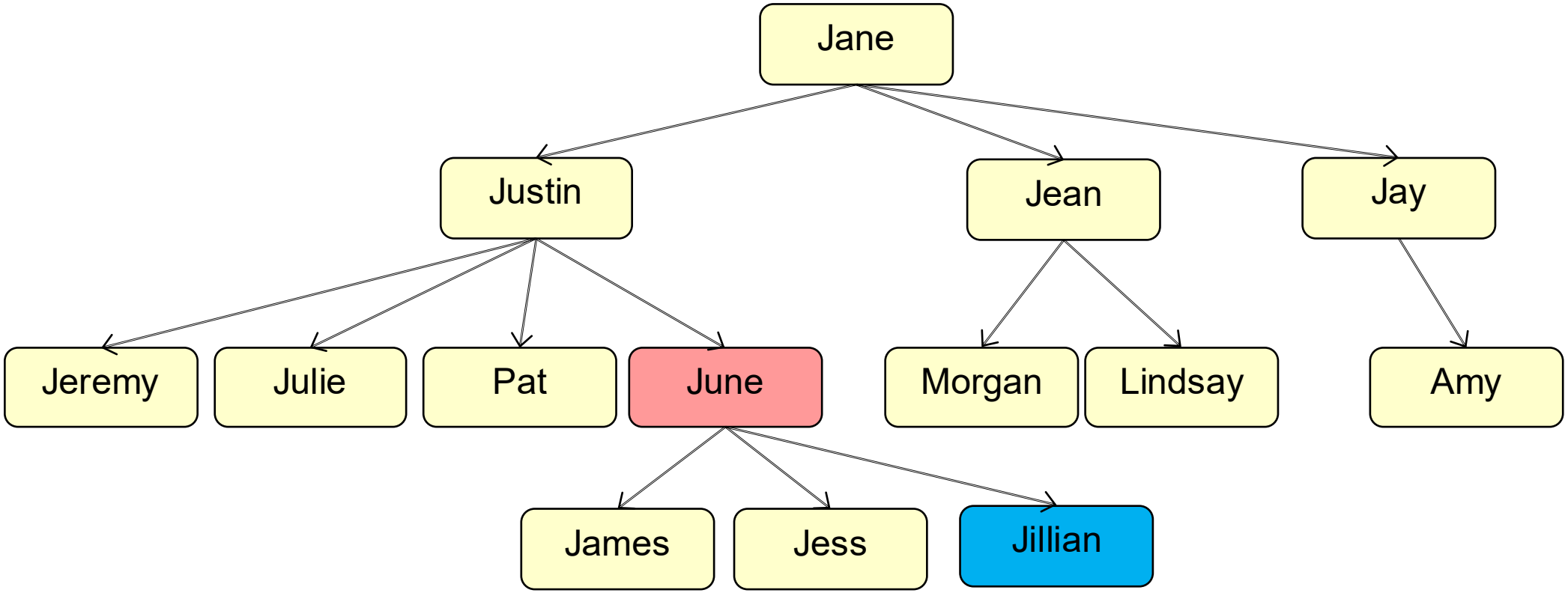
- Traversing the tree by level

```
public void printAll (Person root){
    if (root==null) { return; }
    Queue<Person> todo = new ArrayDequeue<Person>();
    todo.offer(root);
    while (!todo.isEmpty()){
        Person p = todo.poll();
        UI.println(p.getName());
        for (Person child : p.getChildren(){
            todo.offer(child);
        }
    }
}
```



Adding a child to a Tree

- Add a new person as a child of the given parent node
- Have to find the parent node,
  and then add the new Person to that node
- add(Person root, Person ch, String name) add(mytree, Jillian, “June”)



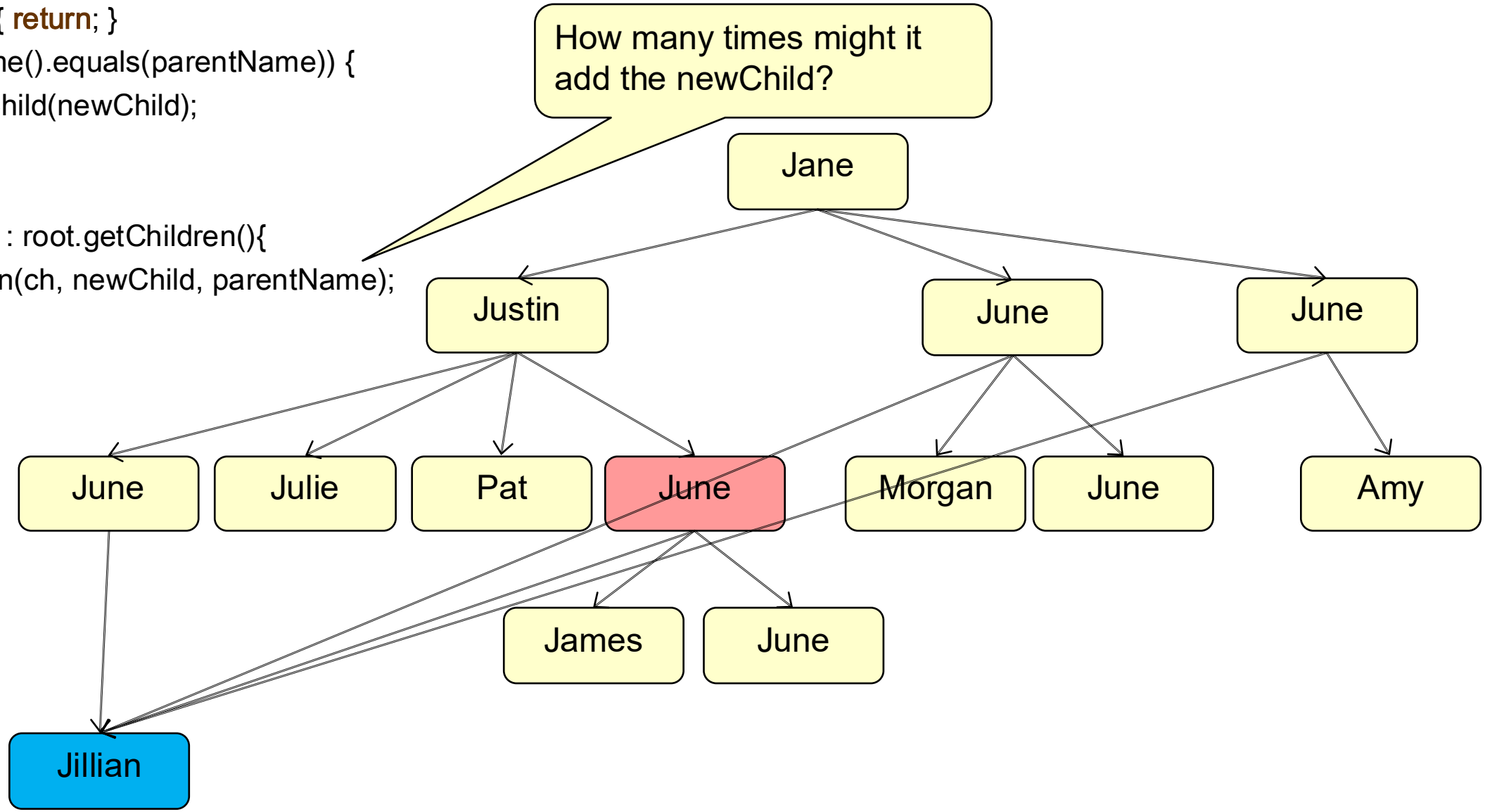
Adding to a node in a Tree

- Depth first traversal

```
public void addPerson(Person root, Person newChild, String parentName){
    if (root==null) { return; }
    if (root.getName().equals(parentName)) {
        root.addChild(newChild);
        return;
    }
    for (Person ch : root.getChildren(){
        addPerson(ch, newChild, parentName);
    }
}
```

Adding a child to a Tree

```
public void addPerson(Person root, Person newChild, String parentName){
    if (root==null) { return; }
    if (root.getName().equals(parentName)) {
        root.addChild(newChild);
        return;
    }
    for (Person ch : root.getChildren(){
        addPerson(ch, newChild, parentName);
    }
}
```



Adding to a node in a Tree

- Depth first traversal – need to exit out ALL levels if added the new child.
- Return a boolean to signal success.

```
public boolean addPerson(Person root, Person newChild, String parentName){
    if (root==null) { return false; }
    if (root.getName().equals(parentName)) {
        root.addChild(newChild);
        return true;
    }
    for (Person ch : root.getChildren(){
        if ( addPerson(ch, newChild, parentName) ) {
            return true;
        }
    }
    return false;
}
```

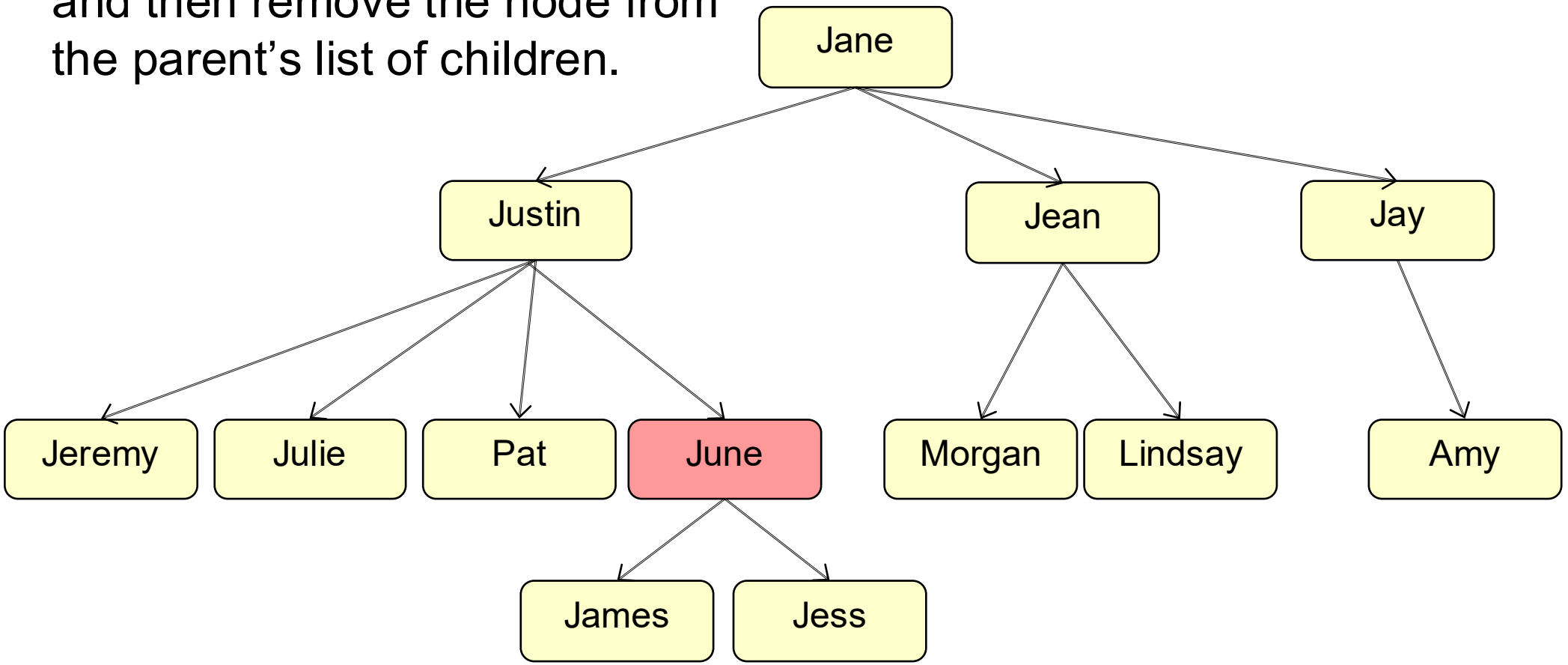
If succeeded here, return success

If succeeded in one subtree, return immediately

Removing a node from a Tree

- If you find a node, how do you remove it?

- Have to find the node's parent, and then remove the node from the parent's list of children.



Removing a node (and subtree) from a Tree

- Depth first traversal with “look ahead”

```
public void removePerson(Person tree, String name){
    if (tree ==null) { return; }
    for (Person child : tree.getChildren(){
        if (child.getName().equals(name)) {
            tree.removeChild(child);
            return;
        }
        else {
            removePerson(child, name);
        }
    }
}
```

How many nodes might it remove?

What if the person to remove is at the root of the tree

Not allowed to remove child from inside the foreach loop!!

Removing a node (and subtree) from a Tree

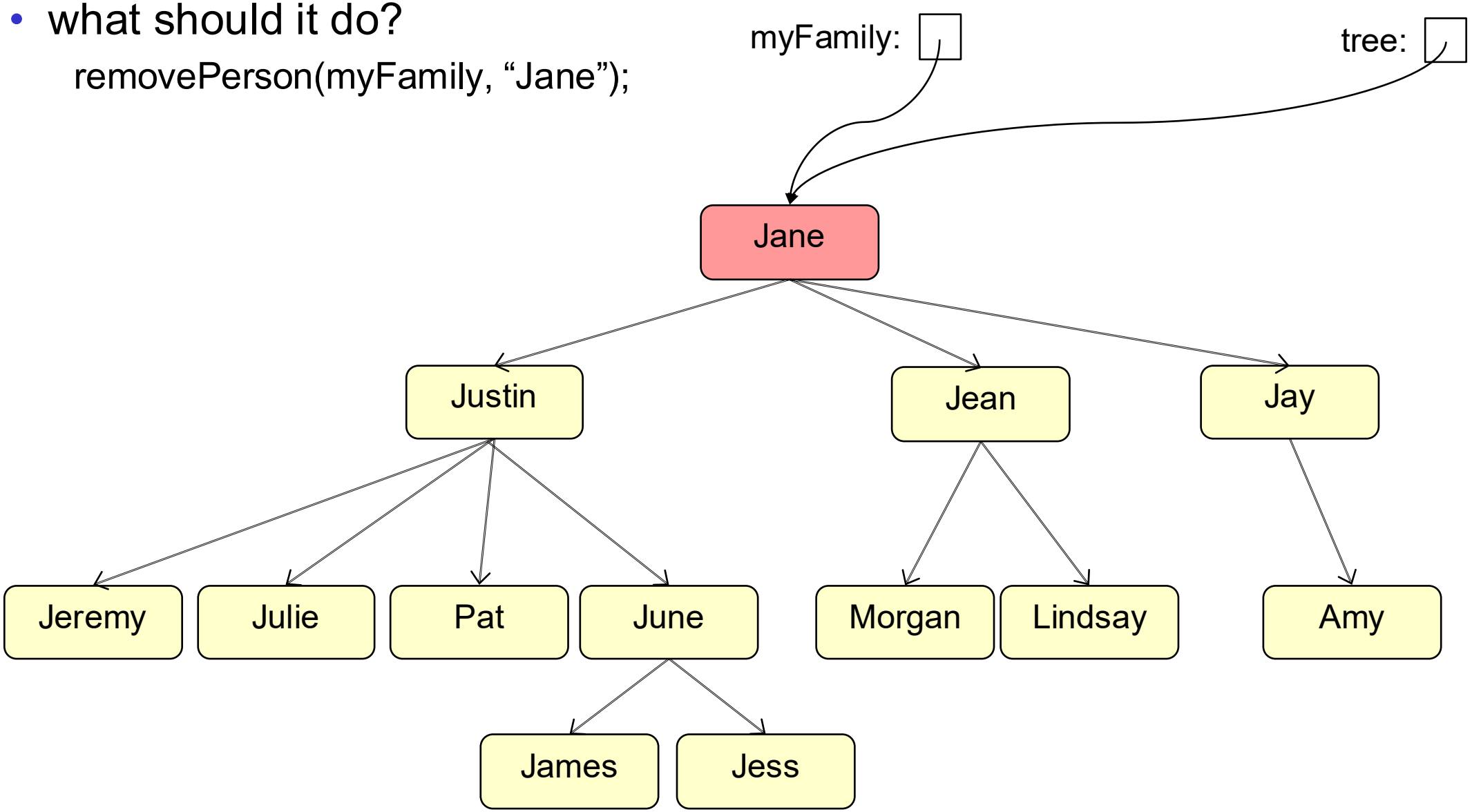
- DF traversal, “look ahead”, remove after loop, return success when removed

```
public boolean removePerson(Person tree, String name){
    if (tree == null){ return false; }
    Person chToRemove = null;
    for (Person ch : tree.getChildren()){
        if (ch.getName().equals(name)){  chToRemove = ch;  break;  }
        else {
            if (removePerson(ch, name)) { return true; }
        }
    }
    if (chToRemove==null){ return false; }
    tree.removeChild(chToRemove);
    return true;
}
```

Removing the root node from a Tree

- what should it do?

  removePerson(myFamily, “Jane”);



General Trees with parent links

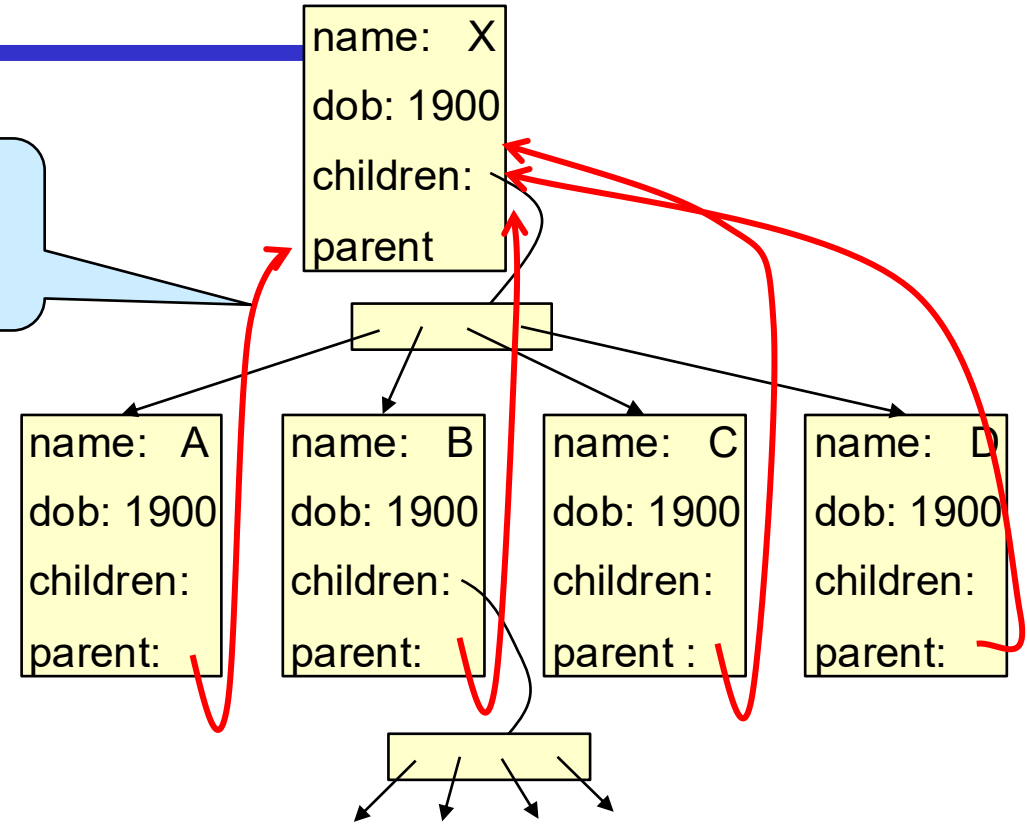
```
public class Person {
    private String name;
    private int dob;
    private Set<Person> children;
    private Person parent;

        :

    public Person getParent(){ return parent; }
    public void setParent(Person p){parent = p; }

    public Set<Person> getChildren(){ return Collections.unmodifiableSet(children);}
    public void addChild(Person ch){ children.add(ch); }
    public void removeChild(Person ch){ children.remove(ch); }
}
```

Parent links make it easy to get the parent of a node

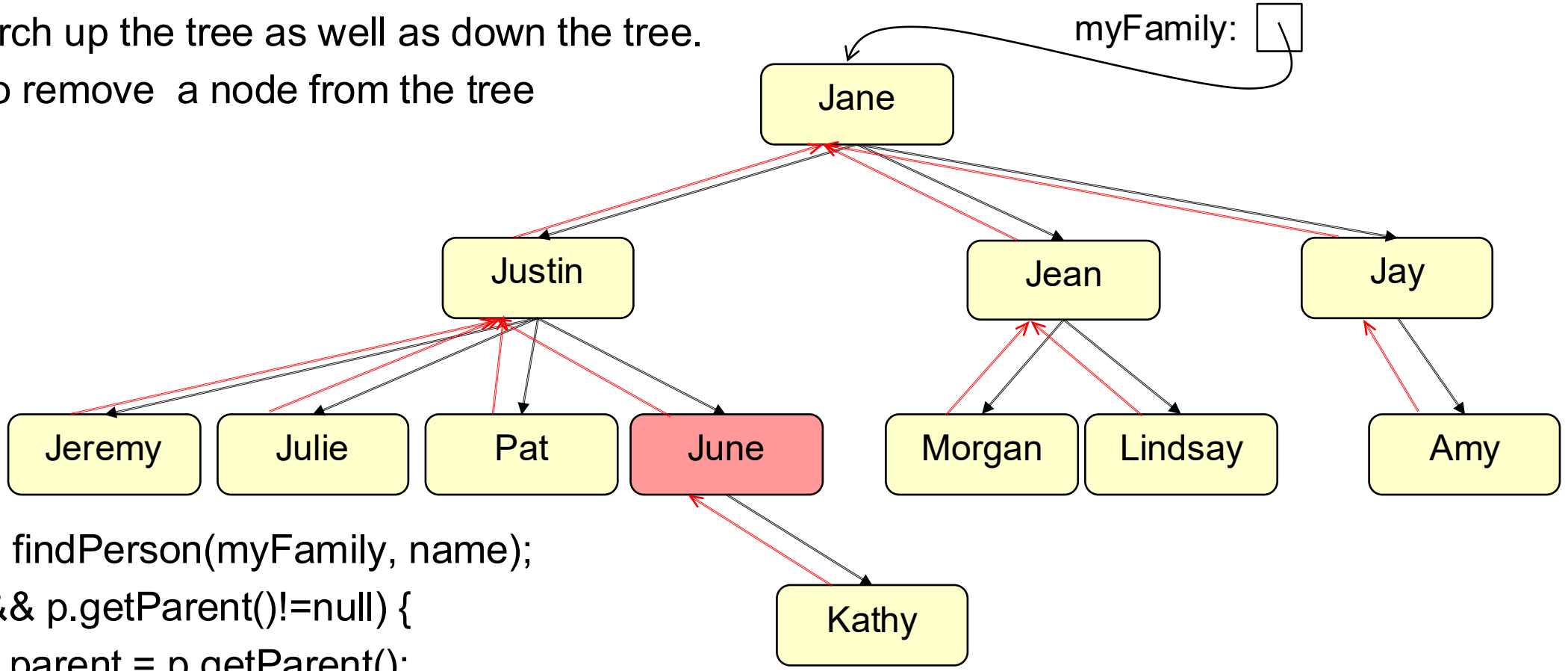


General Trees with parent links

- Each node can have a reference to its parent as well as its children

  ⇒ can search up the tree as well as down the tree.

  ⇒ easier to remove a node from the tree




```
Person p = findPerson(myFamily, name);
if (p!=null && p.getParent()!=null) {
    Person parent = p.getParent();
    parent.removeChild(p);
    p.setParent(null);
}
```

Following parent links: OrganisationChart

To determine horizontal placement of Position node:

- Position contains horizontal **offset** from parent (manager)
  Must find parent's position and then add the offset:

(in Position class)

```
public int getX(){
    if (this.manager==null)  {                    // this is the root of the tree.
        return  BASE_X + this.offset;
    }
    else {                                        // relative to position of parent.
        return this.manager.getX() + this.offset;
    }
}
```

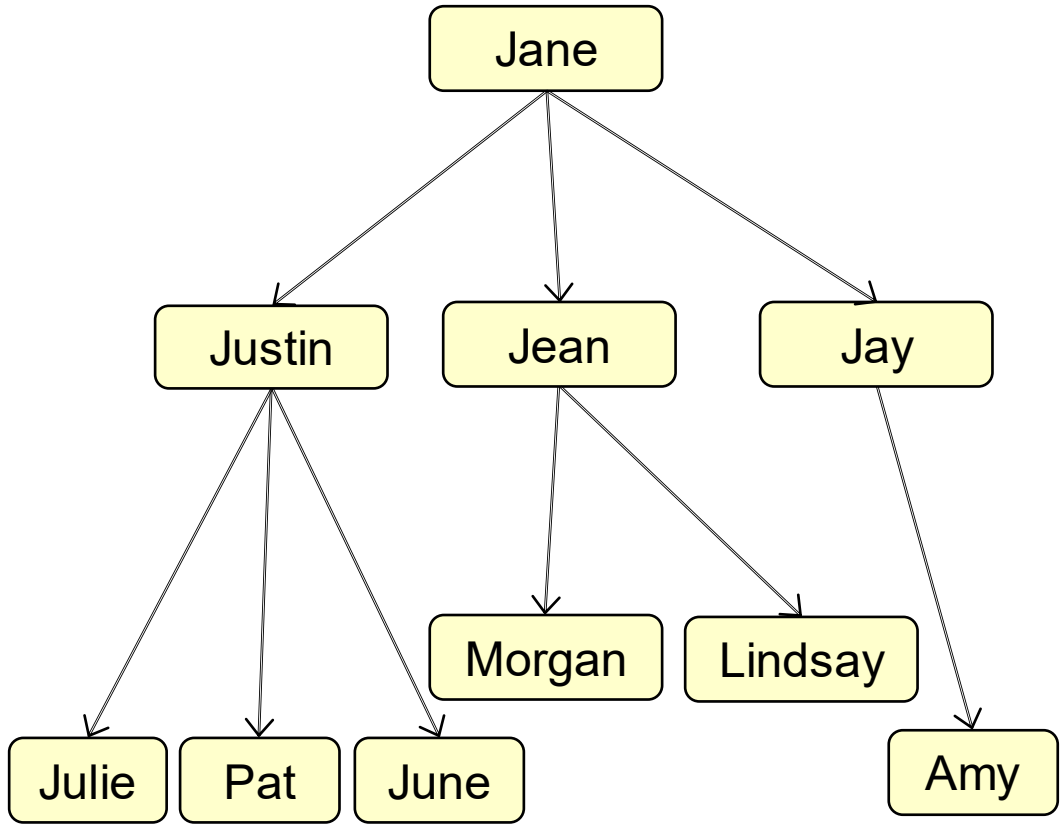
Traversing General Trees

Pre-order:

```
public void printTree(Person p){
    if (p==null) { return;  }
    UI.println(p);
    for (Person child : p.getChildren(){
        printTree(child);
    }
}
```

Post-order:

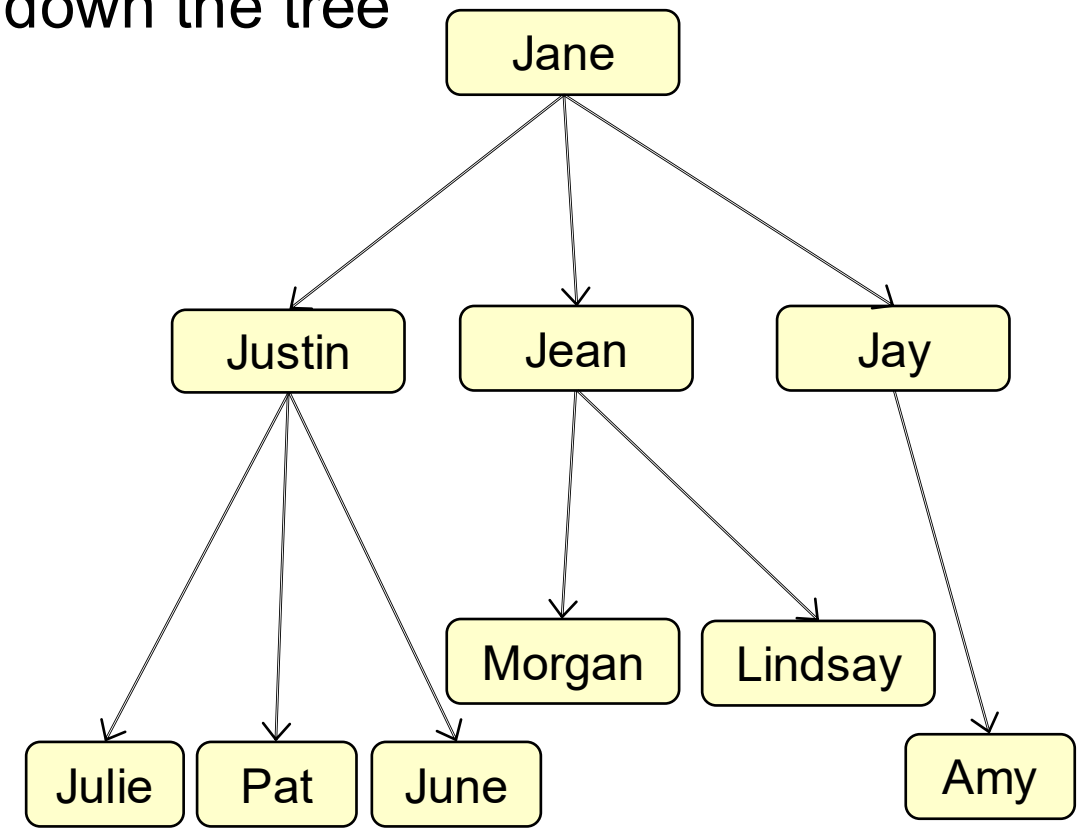
```
public void printTree(Person p, String indent){
    if (p==null)  { return;  }
    for (Person child : p.getChildren(){
        printTree(child, indent+"   ");
    }
    UI.println(indent + p);
}
```



Traversing General Trees

Printing with indentation - passing values down the tree

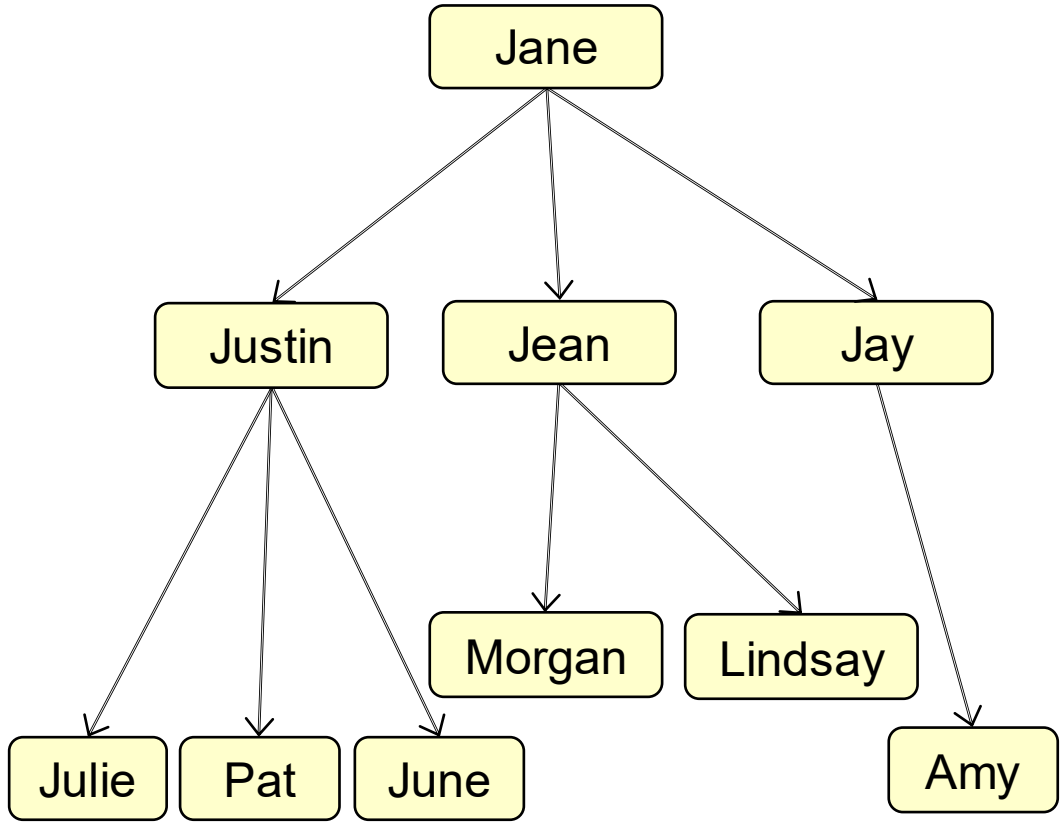
```
public void printTree(Person p, String indent){
    if (p==null) { return; }
    UI.println(indent + p);
    for (Person child : p.getChildren(){
        printTree(child, indent+"  ");
    }
}
```



Traversing General Trees: returning values

```
/** Find a Person in a tree with a specified name, return null if no such Person */

public Person findInTree(String name,  Person tree){
    if (tree==null)  {
        return null;
    }
    if (tree.getName().equals(name))  {
        return tree;
    }
    for (Person child : tree.getChildren(){
        Person ans = findInTree(name, child);
        if (ans != null) {
            return ans;
        }
    }
    return null;
}
```



Traversing General Trees vs Binary Tree

```
import java.util.*;

class GeneralTreeNode {

    int data;

    ArrayList<GeneralTreeNode> children;

    GeneralTreeNode(int data) {

        this.data = data;

        children = new ArrayList<>();
    }
}
```

```
class BinaryTreeNode {
    int data;
    BinaryTreeNode left;
    BinaryTreeNode right;


    BinaryTreeNode(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
```