Implementing Trees

Lecture 22

Menu

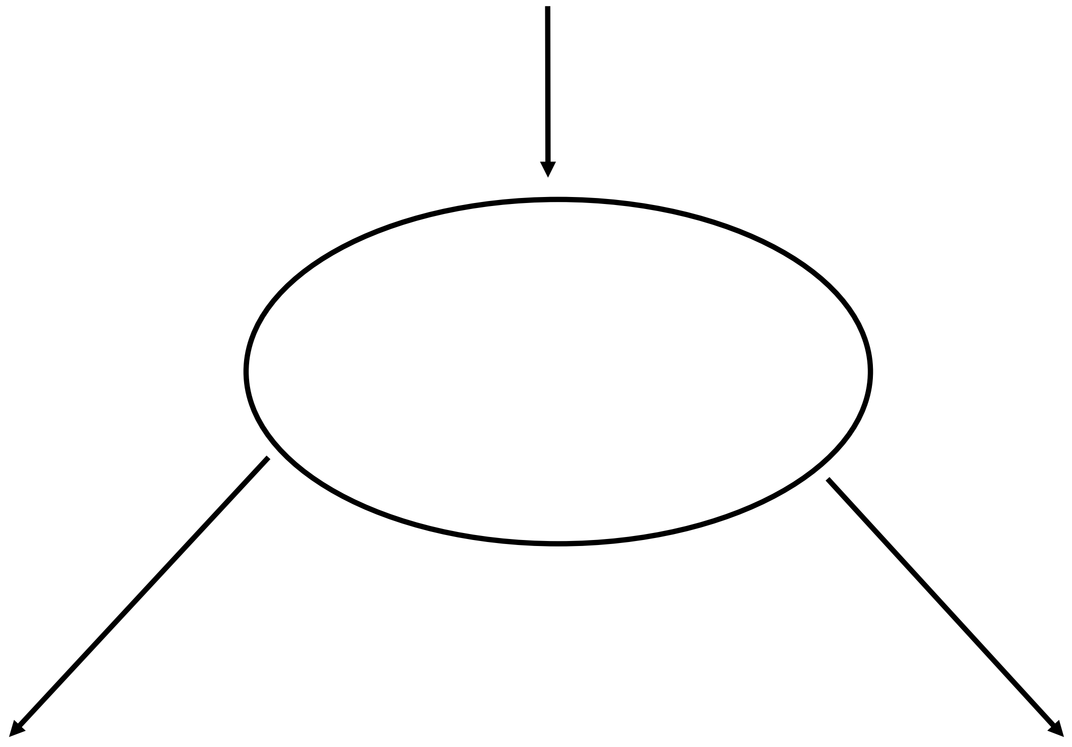
- Abstract Data Type (ADT) for Trees
- Implementing Binary Trees: issues & considerations, data structure?
- Implementing General Trees: issues & considerations, data structure?

- Applications for Trees
- Tree Traversal

Binary Tree ADT

```
public interface BinaryTree<T> {
    BinaryTree<T> BinaryTree(T value);

    BinaryTree<T> getLeft();
    BinaryTree<T> getRight();
    void setLeft(BinaryTree<T> subtree);
    void setRight(BinaryTree<T> subtree);

    BinaryTree<T> find(T value);
    boolean contains(T value);

    boolean isEmpty();
    boolean isLeaf();
    int size();
    }
```

What does the Binary Tree ADT/class need?



Binary Tree

- Each node contains a value
- Each node has a left child and a right child (may be null)

```
public class BinaryTree<V> {

    private V value;
    private BinaryTree<V> left;
    private BinaryTree<V> right;

    /**Creates a new tree with one node
        (a root node) containing the specified value  */
    public BinaryTree(V value) {
      this.value = value;
    }
```

Get and Set

```
public V getValue() {
    return value;
 }
 public BinaryTree<V> getLeft() {
    return left;
 }
 public BinaryTree<V> getRight() {
    return right;
 }
 public void setValue(V val) {
     value = val;
 }
 public void setLeft(BinaryTree<V> tree) {
     left = tree;
 }
 public void setRight(BinaryTree<V> tree) {
     right = tree;
 }
```

isLeaf and find

```
  public boolean isLeaf() {
      return (right == null) && (left == null);
   }

   public BinaryTree<V> find(V val) {
      if (value.equals(val))      return this;
      if (left != null) {
          BinaryTree<V> ans = left.find(val);
          if (ans != null)          return ans;
      }
      if (right != null) {
          BinaryTree<V> ans = right.find(val);
          if (ans != null)          return ans;
      }
      return null;
   }
}
```

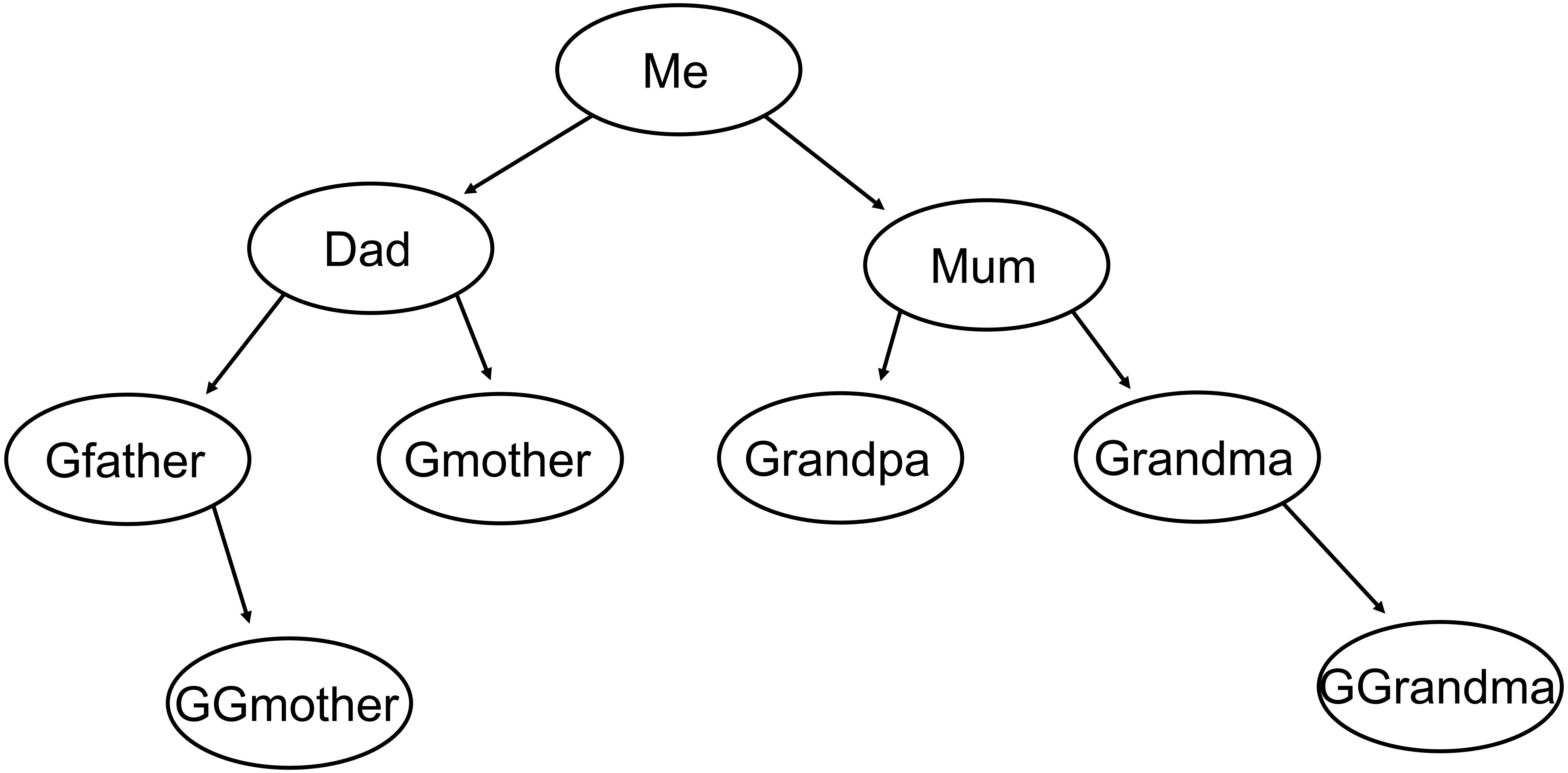
Using BinaryTree

- A method that constructs a tree

```
public static void main(String[] args){
  BinaryTree<String> myTree = new BinaryTree<String>("Me");
  myTree.setLeft(new BinaryTree<String>("Dad"));
  myTree.setRight(new BinaryTree<String>("Mum"));
  myTree.getLeft().setLeft(new BinaryTree<String>("Gfather"));
  myTree.getLeft().setRight(new BinaryTree<String>("Gmother"));
  myTree.getRight().setLeft(new BinaryTree<String>("Grandpa"));
  myTree.getRight().setRight(new BinaryTree<String>("Grandma"));
  myTree.getRight().getRight().setRight(
         new BinaryTree<String>("GGrandma"));

  BinaryTree<String> gf = myTree.find("Gfather");
  if (gf!=null)
     gf.setRight(new BinaryTree<String>(“GGmother"));
}
```

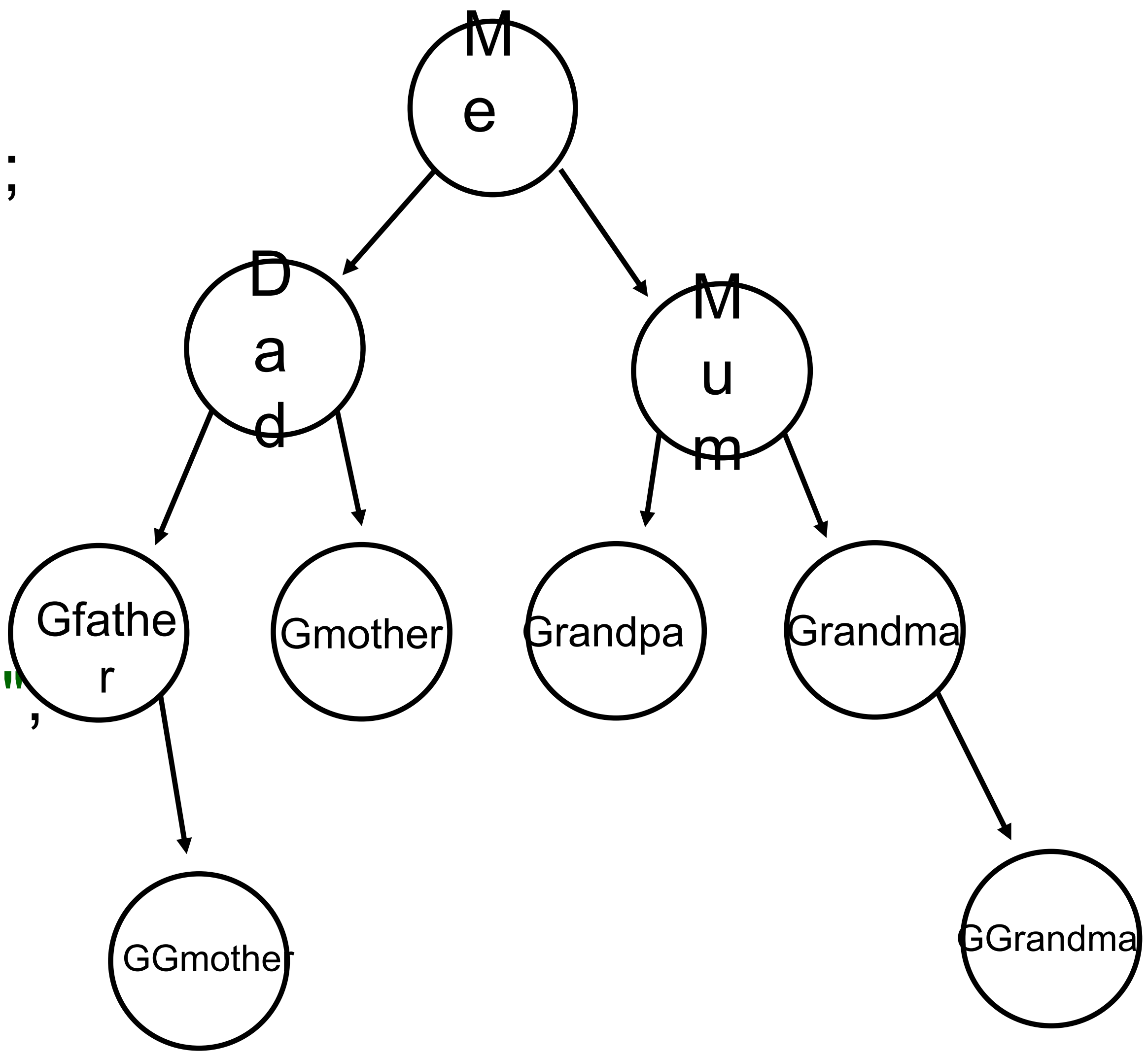
Ex: Show the final tree contents after the above program execution.

myTree


Me
Dad
Mum
Gfather
Gmother
Grandpa
Grandma
GGmother
GGrandma

Using BinaryTree

```
BinaryTree<String> ma = myTree;
   while(ma.getRight()!=null)   ma = ma.getRight();

BinaryTree<String> pa = myTree;
    while(pa.getLeft()!=null)   pa = pa.getLeft();

System.out.format(
      "paternal ansc = %s, maternal ansc = %s\n\n",
      pa.getValue(), ma.getValue());
```



What would be the results of execution?

paternal ans = Gfather, maternal ans = GGrandma

Using BinaryTree

```
public static void printAll(BinaryTree<String> tree, String indent){
    System.out.println(indent+ tree.getValue());
    if (tree.getLeft()!=null)
      printAll(tree.getLeft(), indent+"  " );
    if (tree.getRight()!=null)
      printAll(tree.getRight(), indent+"  " );
}
```

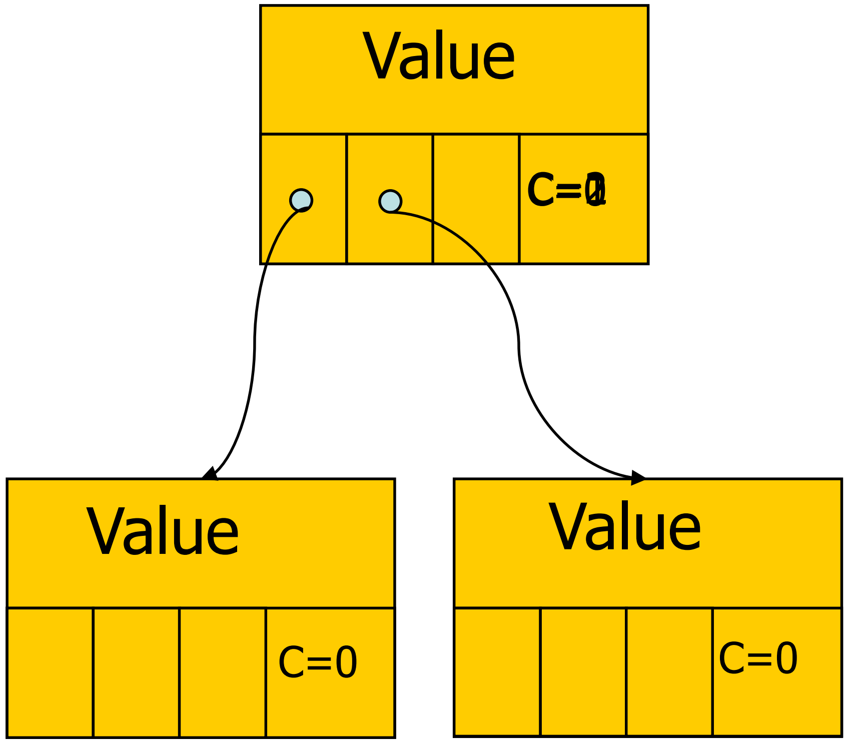
What traversal scheme is this?

PreOrder traversal

General Tree

- A node in the tree can have any number of children
- We keep the children in the order that they were added.

```
public class GeneralTree<V> {

    private V value;
    private List< GeneralTree<V> > children;

    public GeneralTree(V value) {
        this.value = value;
        this.children = new ArrayList< GeneralTree<V> >();
    }
...
}
```



Get

- Children are ordered, so can ask for the *i* th child

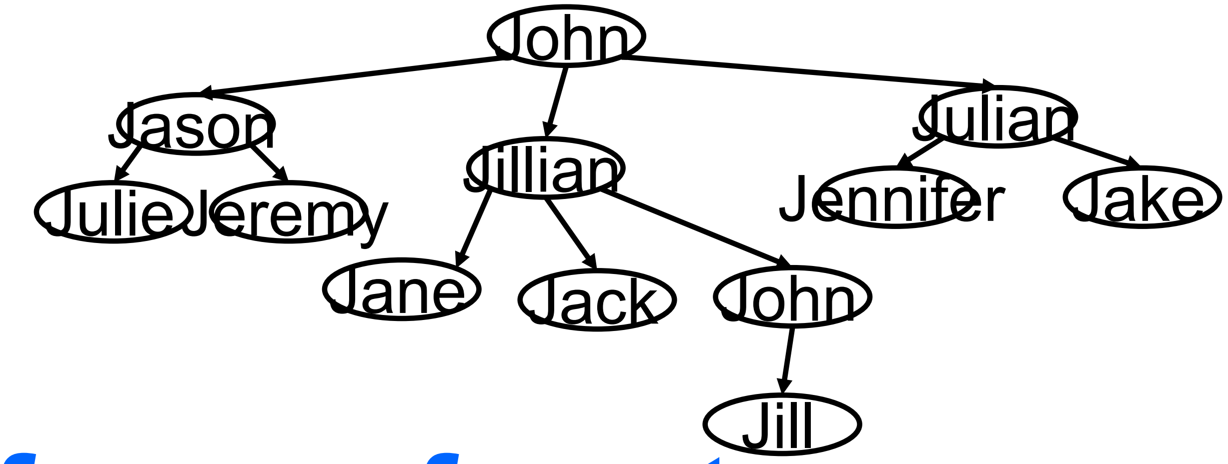
```
public V getValue() {
    return value;
}

public List< GeneralTree<V> > getChildren() {
    return children;
}

public GeneralTree<V> getChild(int i) {
    if (i>=0 && i < children.size())
        return children.get(i);
    else
        return null;
}
```

add and find

```
public void addChild(GeneralTree<V> child) {
    children.add(child);
 }
 public void addChild(int i, GeneralTree<V> child) {
    children.add(i, child);
 }

public GeneralTree<V> find(V val) {
    if (value.equals(val))       return this;
    for(GeneralTree<V> child : children){
        GeneralTree<V> ans = child.find(val);
         if (ans != null)           return ans;
     }
    return null;
 }
```

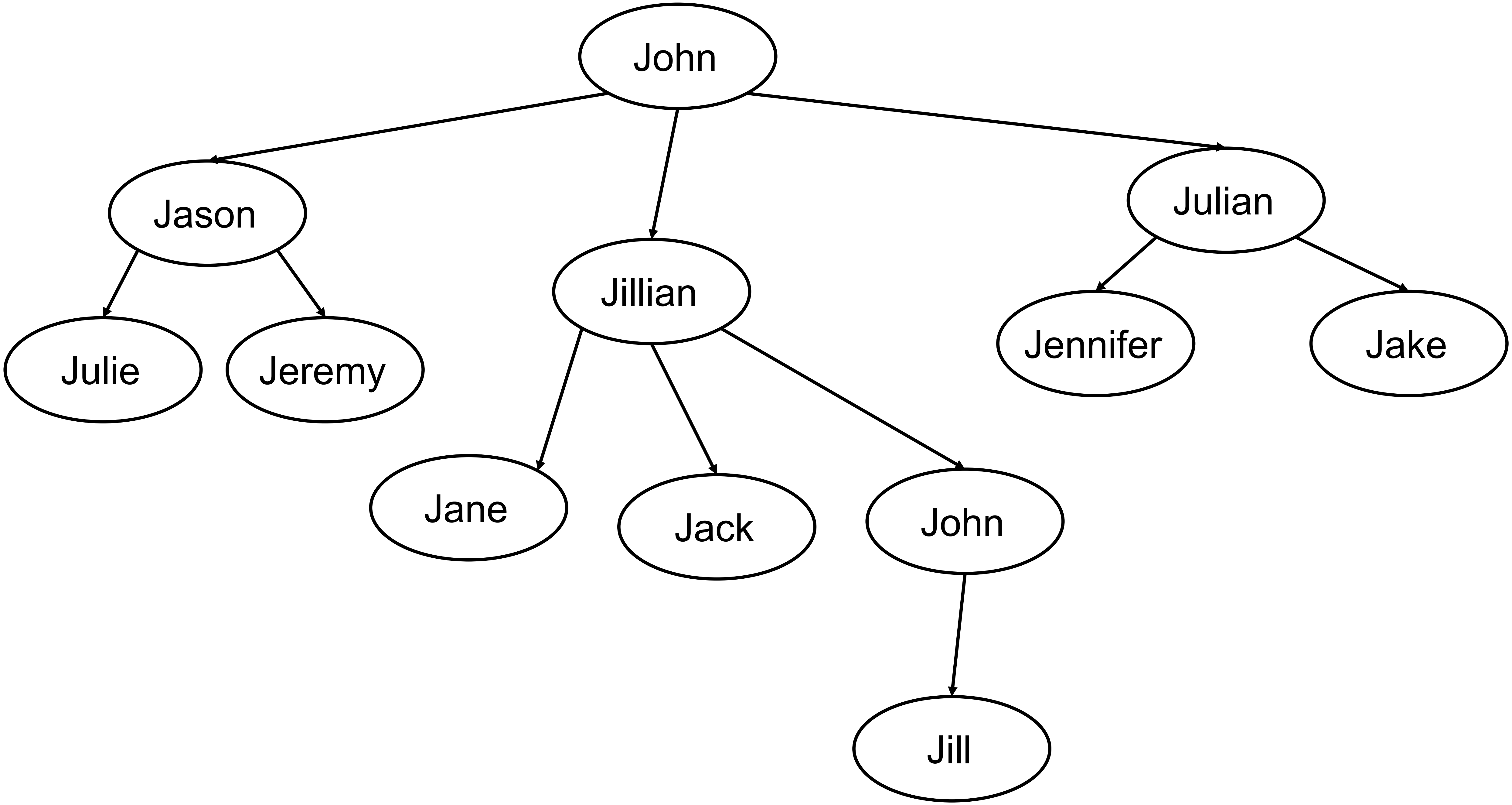
Using General Tree

```
public static void main(String[] args){
 GeneralTree<String> mytree = new GeneralTree<String>("John");
    mytree.addChild(new GeneralTree<String>("Jason"));
    mytree.addChild(new GeneralTree<String>("Jillian"));
    mytree.addChild(new GeneralTree<String>("Julian"));
    mytree.getChildren().get(0).addChild(new GeneralTree<String>("Julie"));
    mytree.getChildren().get(0).addChild(new GeneralTree<String>("Jeremy"));
    mytree.getChildren().get(1).addChild(new GeneralTree<String>("Jane"));
    mytree.getChildren().get(1).addChild(new GeneralTree<String>("Jack"));
    mytree.getChildren().get(1).addChild(new GeneralTree<String>("John"));
    mytree.getChildren().get(2).addChild(new GeneralTree<String>("Jennifer"));
    mytree.getChildren().get(2).addChild(new GeneralTree<String>("Jake"));
    mytree.getChildren().get(1).getChildren().get(2).addChild(new
            GeneralTree<String>("Jill"));
}
```



Ex. Draw the final form of mytree

The Tree

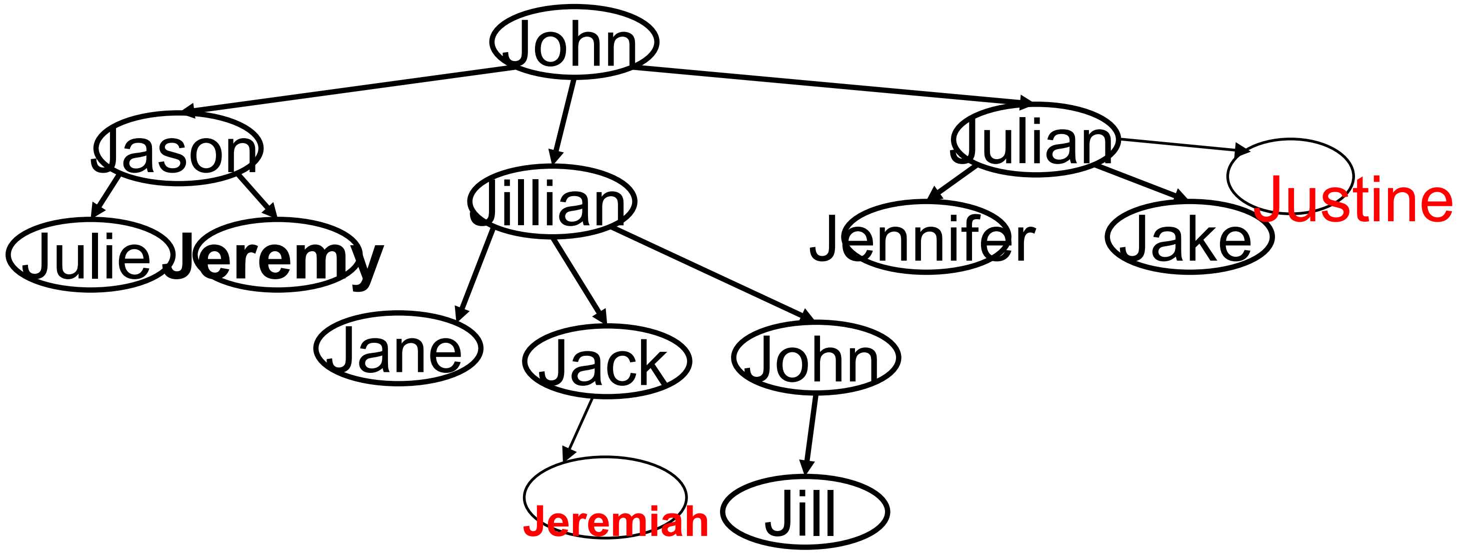

John
Jason
Jillian
Julian
Julie
Jeremy
Jennifer
Jake
Jane
Jack
John
Jill

More Adding

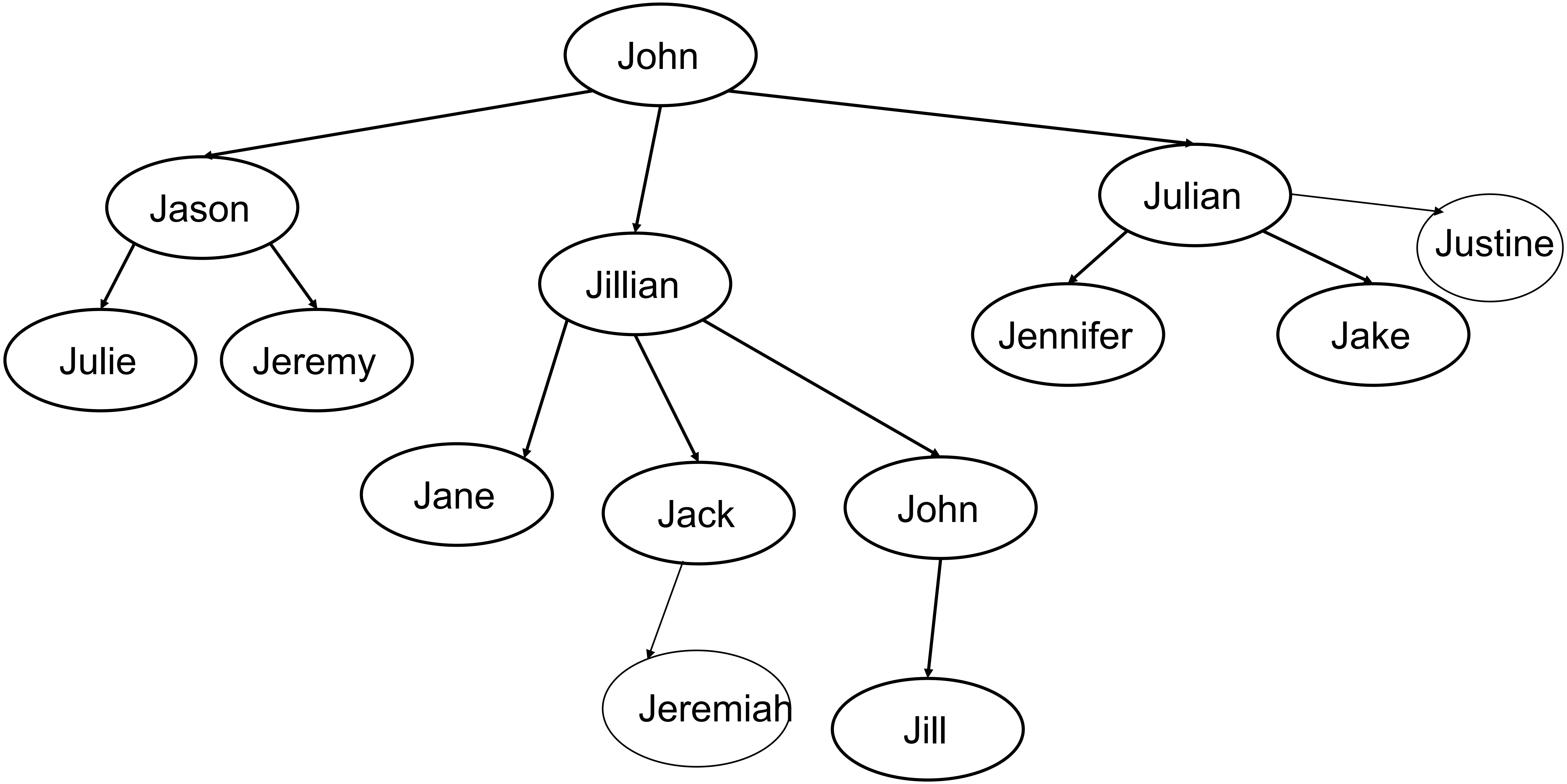
```
GeneralTree<String> thrd = mytree.getChildren().get(2);
thrd.addChild(new GeneralTree<String>("Justine"));

GeneralTree<String> gc = mytree.find("Jack");
if (gc!=null)
     gc.addChild(new GeneralTree<String>("Jeremiah"));

System.out.println ("2nd child of 1st child: "+
                    mytree.getChild(0).getChild(1).getValue());
```

Jeremy

mytree = ?



The Tree


John
Jason
Jillian
Julian
Justine
Julie
Jeremy
Jennifer
Jake
Jane
Jack
John
Jeremiah
Jill

printAll

```
printAll(mytree, "");

public static void printAll(GeneralTree<String> tree, String indent){
     System.out.println(indent+ tree.getValue());
     for(GeneralTree<String> child : tree.getChildren())
       printAll(child, indent+"  " );
   }
```

What traversal scheme is this?

PreOrder traversal

Summary

- Investigated the design and implementation of binary tree and general tree
  - Issues
  - Data Structure

- Tree class design and implementation is quite simple… if you get the ideas of recursion
- Every algorithm can be expressed iteratively or recursively
- One way is usually much better than the other!
  - In trees, recursion is normally nicer than iteration (but not always)